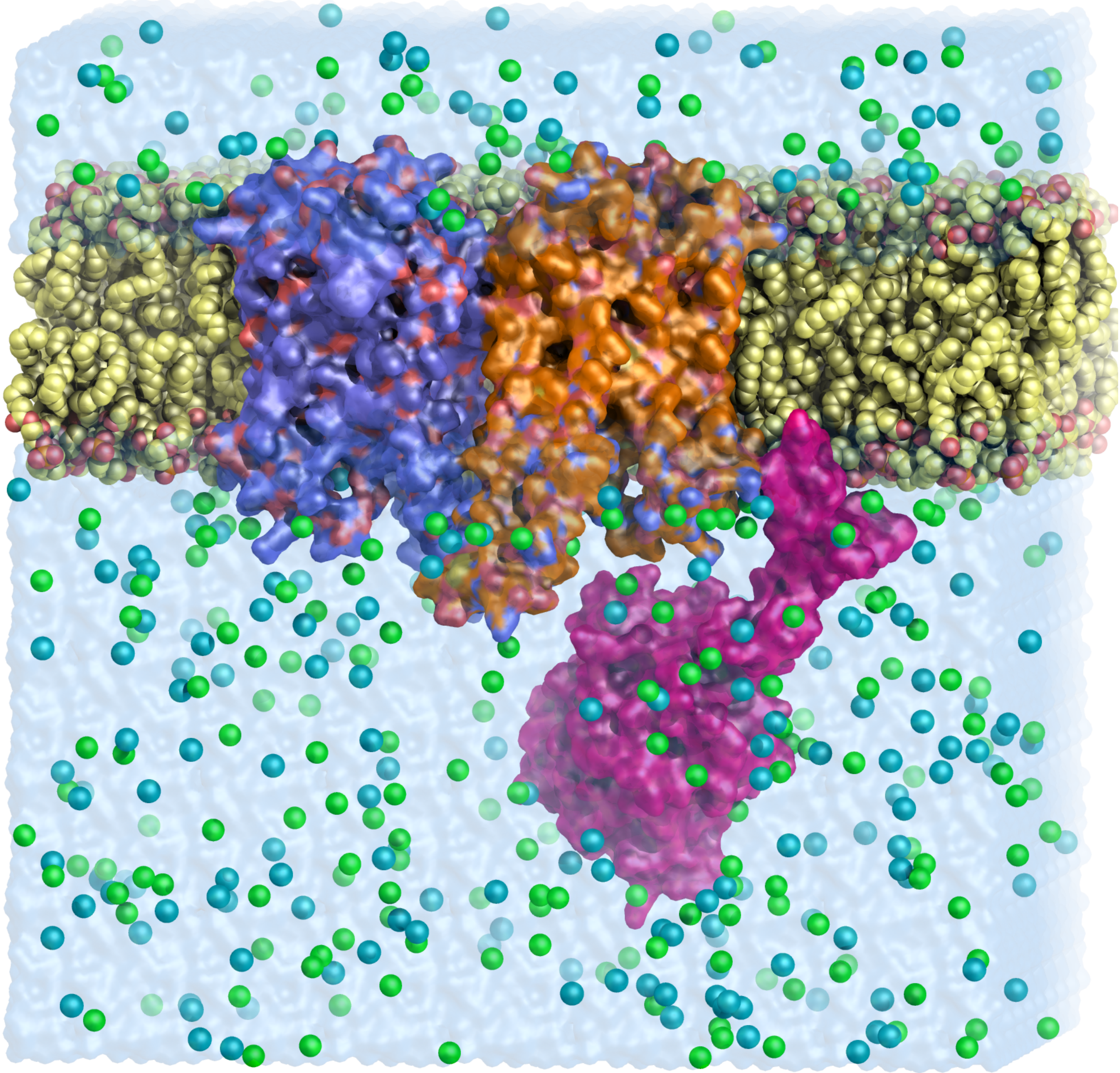
Amber 2017
Reference Manual

(Covers Amber16 and AmberTools17)



Amber 2017
Reference Manual
(Covers Amber16 and AmberTools17)

Principal contributors to the current codes:

David A. Case (Rutgers)
Ross C. Walker (UCSD, GSK)
Thomas E. Cheatham III (Utah)
Carlos Simmerling (Stony Brook)
Adrian Roitberg (Florida)
Kenneth M. Merz (Michigan State)
Ray Luo (UC Irvine)
Tom Darden (OpenEye)
Junmei Wang (Pitt)
Robert E. Duke (NIEHS)
Daniel R. Roe (Utah, NIH)
Scott LeGrand (Amazon)
Jason Swails (Lutron)
Andreas W. Götz (UCSD)
David Cerutti (Rutgers)
Tyler Luchko (CSU Northridge)
Gérald Monard (U. Lorraine)
Celeste Sagui (NCSU)
Feng Pan (NCSU)
Charles Lin (UCSD)
Daniel Mermelstein (UCSD)
Pengfei Li (Michigan State, UIUC)
Alexey Onufriev (Virginia Tech)
Saeed Izadi (Virginia Tech, Genentech)
Romain M. Wolf (Novartis)
Xiongwu Wu (NIH)
Holger Gohlke (Düsseldorf)
Nadine Homeyer (Düsseldorf)
Ruxi Qi (UC Irvine)
Wesley M. Botello-Smith (UC Irvine)
Li Xiao (UC Irvine)
D'Artagnan Greene (UC Irvine)
Tim Giese (Rutgers)
Taisung Lee (Rutgers)
Darrin York (Rutgers)
Jian Liu (Peking Univ.)
Hai Nguyen (Stony Brook, Rutgers)
Igor Omelyan (NINT)
Andriy Kovalenko (NINT)
Peter A. Kollman (UC San Francisco)

For more information, please visit *http://ambermd.org/contributors.html*

Acknowledgments

Research support from DARPA, NIH, ONR, DOE and NSF is gratefully acknowledged, along with support from NVIDIA, Amazon and Exxact. Many people helped add features to various codes; these contributions are described in the documentation for the individual programs; see also *http://ambermd.org/contributors.html*.

Recommended Citation:

- When citing Amber 2017 (comprised of AmberTools17 and Amber16) in the literature, the following citation should be used:

  D.A. Case, D.S. Cerutti, T.E. Cheatham, III, T.A. Darden, R.E. Duke, T.J. Giese, H. Gohlke, A.W. Goetz, D. Greene, N. Homeyer, S. Izadi, A. Kovalenko, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, D. Mermelstein, K.M. Merz, G. Monard, H. Nguyen, I. Omelyan, A. Onufriev, F. Pan, R. Qi, D.R. Roe, A. Roitberg, C. Sagui, C.L. Simmerling, W.M. Botello-Smith, J. Swails, R.C. Walker, J. Wang, R.M. Wolf, X. Wu, L. Xiao, D.M. York and P.A. Kollman (2017), AMBER 2017, University of California, San Francisco.

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

Notes

- We thank Chris Bayly and Merck-Frosst, Canada for permission to include charge increments for the AM1-BCC charge scheme.
- Some of the force field routines in NAB were adapted from similar routines in the MOIL program package: R. Elber, A. Roitberg, C. Simmerling, R. Goldstein, H. Li, G. Verkhivker, C. Keasar, J. Zhang and A. Ulitsky, "MOIL: A program for simulations of macromolecules" *Comp. Phys. Commun.* **91,** 159-189 (1995).

Cover illustration: Dimer of the G-protein coupled receptor TGR5 (blue and orange surface representations) interacting with a green fluorescent protein (magenta) attached to the TGR5 C-terminus. Figure prepared by Christoph Gertzen and Holger Gohlke, based on work reported by A. Greife, S. Felekyan, Q. Ma, C.G.W. Gertzen, L. Spomer, M. Dimura, T.O. Peulen, C. Wöhler, D. Häussinger, H. Gohlke, V. Keitel and C.A.M. Seidel, Structural assemblies of the di- and oligomeric G-protein coupled receptor TGR5 in live cells: an MFIS-FRET and integrative modelling study. *Sci. Rep.* **6,** 36792 (2016).

Contents

Contents 5

I. Introduction and Installation 13

1. Introduction 15
- 1.1. Information flow in Amber 15
- 1.2. List of programs 18

2. Installation 23
- 2.1. Uninstalling and cleaning 26
- 2.2. Python in Amber 26
- 2.3. Applying Updates 28
- 2.4. Contacting the developers 30

II. Amber force fields 31

3. Molecular mechanics force fields 33
- 3.1. Proteins 34
- 3.2. Nucleic acids 37
- 3.3. Carbohydrates 39
- 3.4. Lipids 47
- 3.5. Solvents 49
- 3.6. Ions 50
- 3.7. Modified amino acids and nucleotides 52
- 3.8. Force fields related to semi-empirical QM 52
- 3.9. Obsolete force field files 53

4. The Generalized Born/Surface Area Model 57
- 4.1. GB/SA input parameters 59
- 4.2. ALPB (Analytical Linearized Poisson-Boltzmann) 62

5. GBNSR6 64
- 5.1. GB equations available in gbnsr6 64
- 5.2. Numerical implementation of the R6 integral 64
- 5.3. Usage 65

6. PBSA 68
- 6.1. Introduction 68
- 6.2. Usage and keywords 71
- 6.3. Example inputs and demonstrations of functionalities 80
- 6.4. Visualization functions in *pbsa* 82
- 6.5. *pbsa* in *sander* and NAB 89
- 6.6. GPU accelerated *pbsa* 91

7. Reference Interaction Site Model **94**
7.1. Introduction . . . 94
7.2. Practical Considerations . . . 99
7.3. Work Flow . . . 100
7.4. rism1d . . . 102
7.5. 3D-RISM in `NAB` . . . 105
7.6. `rism3d.snglpnt` . . . 107
7.7. 3D-RISM in sander . . . 110
7.8. RISM File Formats . . . 118

8. Empirical Valence Bond **124**
8.1. Introduction . . . 124
8.2. General usage description . . . 125
8.3. Biased sampling . . . 127
8.4. Quantization of nuclear degrees of freedom . . . 130
8.5. Distributed Gaussian EVB . . . 130
8.6. EVB input variables and interdependencies . . . 132

9. sqm: Semi-empirical quantum chemistry **137**
9.1. Available Hamiltonians . . . 137
9.2. Dispersion and hydrogen bond correction . . . 139
9.3. Usage . . . 140

10. QM/MM calculations **146**
10.1. Built-in semiempirical NDDO methods and SCC-DFTB . . . 146
10.2. Interface for *ab initio* and DFT methods . . . 155
10.3. Adaptive solvent QM/MM simulations . . . 170
10.4. Adaptive buffered force-mixing QM/MM . . . 175
10.5. SEBOMD: SemiEmpirical Born-Oppenheimer Molecular Dynamics . . . 182

11. paramfit **187**
11.1. Usage . . . 188
11.2. The Job Control File . . . 189
11.3. Multiple molecule fits . . . 195
11.4. Fitting Forces . . . 195
11.5. Examples . . . 196

III. System preparation **198**

12. Preparing PDB Files **200**
12.1. Cleaning up Protein PDB Files for AMBER . . . 200
12.2. Residue naming conventions . . . 201
12.3. Chains, Residue Numbering, Missing Residues . . . 202
12.4. pdb4amber . . . 202
12.5. reduce . . . 205

13. LEaP **206**
13.1. Introduction . . . 206
13.2. Concepts . . . 206
13.3. Running LEaP . . . 210
13.4. Basic instructions for using LEaP to build molecules . . . 215
13.5. Commands . . . 216
13.6. Building oligosaccharides, lipids and glycoproteins . . . 233

14. Reading and modifying Amber parameter files 242
14.1. Understanding Amber parameter files 242
14.2. ParmEd 250

15. Antechamber and GAFF 278
15.1. Principal programs 278
15.2. A simple example for antechamber 283
15.3. Using the components.cif file from the PDB 286
15.4. Programs called by antechamber 286
15.5. Miscellaneous programs 290
15.6. New Development of Antechamber And GAFF 293
15.7. Metal Center Parameter Builder (MCPB) 294
15.8. Python Metal Site Modeling Toolbox (pyMSMT) 295

16. Setting up crystal simulations 307
16.1. UnitCell 307
16.2. PropPDB 307
16.3. AddToBox 307
16.4. ChBox 309

IV. Running simulations 310

17. sander 312
17.1. Introduction 312
17.2. File usage 313
17.3. Example input files 314
17.4. Namelist Input Syntax 315
17.5. Overview of the information in the input file 316
17.6. General minimization and dynamics parameters 316
17.7. Potential function parameters 327
17.8. Varying conditions 333
17.9. File redirection commands 337
17.10Getting debugging information 337
17.11multisander (and multipmemd) 340
17.12APBS as an alternate PB solver in Sander 341
17.13Programmer's Corner: The sander API 343

18. pmemd 364
18.1. Introduction 364
18.2. Functionality 364
18.3. PMEMD-specific namelist variables 366
18.4. Slightly changed functionality 367
18.5. Parallel performance tuning and hints 368
18.6. GPU Accelerated PMEMD 368
18.7. Intel® Many Integrated Core Architecture 374

19. Atom and Residue Selections 380
19.1. Amber Masks 380
19.2. "Atom Expressions" in NAB Applications 383
19.3. GROUP Specification 383

20. Sampling configuration space 387
20.1. Self-Guided Langevin dynamics 387

20.2. Accelerated Molecular Dynamics . . . 390
20.3. Gaussian Accelerated Molecular Dynamics . . . 393
20.4. Targeted MD . . . 396
20.5. Multiply-Targeted MD (MTMD) . . . 397
20.6. Nudged elastic band calculations . . . 398
20.7. Low-MODe (LMOD) methods . . . 402

21. Free energies **406**
21.1. Thermodynamic integration . . . 406
21.2. Absolute Free Energies using EMIL . . . 415
21.3. Linear Interaction Energies . . . 419
21.4. Umbrella sampling . . . 420
21.5. Replica Exchange Molecular Dynamics (REMD) . . . 421
21.6. Adaptively Biased MD, Steered MD, Umbrella Sampling with REMD and String Method . . . 438
21.7. Steered Molecular Dynamics (SMD) and the Jarzynski Relationship . . . 456

22. Constant pH calculations **458**
22.1. Background . . . 458
22.2. Preparing a system for constant pH . . . 458
22.3. Running at constant pH . . . 460
22.4. Analyzing constant pH simulations . . . 463
22.5. Extending constant pH to additional titratable groups . . . 465
22.6. Constant pH MD Replica Exchange . . . 466
22.7. cphstats . . . 466

23. NMR, X-ray, and cryo-EM/ET refinement **474**
23.1. Distance, angle and torsional restraints . . . 475
23.2. NOESY volume restraints . . . 480
23.3. Chemical shift restraints . . . 481
23.4. Pseudocontact shift restraints . . . 482
23.5. Direct dipolar coupling restraints . . . 484
23.6. Residual CSA or pseudo-CSA restraints . . . 486
23.7. Preparing restraint files for Sander . . . 486
23.8. Getting summaries of NMR violations . . . 493
23.9. Time-averaged restraints . . . 494
23.10Multiple copies refinement using LES . . . 495
23.11Some sample input files . . . 495
23.12X-ray Crystallography Refinement using SANDER . . . 499
23.13EMAP restraints for rigid and flexible fitting into EM maps . . . 500

24. LES **502**
24.1. Preparing to use LES with Amber . . . 502
24.2. Using the ADDLES program . . . 503
24.3. More information on the ADDLES commands and options . . . 505
24.4. Using the new topology/coordinate files with SANDER . . . 506
24.5. Using LES with the Generalized Born solvation model . . . 507
24.6. Case studies: Examples of application of LES . . . 507

25. Quantum dynamics **511**
25.1. Path-Integral Molecular Dynamics . . . 511
25.2. Centroid Molecular Dynamics (CMD) . . . 515
25.3. Ring Polymer Molecular Dynamics (RPMD) . . . 518
25.4. Linearized semiclassical initial value representation . . . 518

25.5. Reactive Dynamics . . . 523
25.6. Isotope effects . . . 526

26. mdgx **531**
26.1. Input and Output . . . 531
26.2. Installation . . . 532
26.3. Special Algorithmic Features of mdgx . . . 532
26.4. Customizable Virtual Site Support in mdgx . . . 533
26.5. Implicitly Polarized Charge Development in mdgx . . . 536
26.6. Restrained Electrostatic Potential Fitting in mdgx . . . 538
26.7. Bonded Term Fitting in mdgx . . . 540
26.8. Configuration Sampling . . . 542
26.9. Thermodynamic Integration . . . 546
26.10Future Directions and Goals of the mdgx Project . . . 546

V. Analysis of simulations **547**

27. mdout_analyzer.py and ambpdb **549**
27.1. ambpdb . . . 549

28. cpptraj **551**
28.1. Running Cpptraj . . . 552
28.2. General Concepts . . . 556
28.3. Data Sets and Data Files . . . 558
28.4. Data File Options . . . 561
28.5. Coordinates (COORDS) Data Set Commands . . . 564
28.6. General Commands . . . 568
28.7. Topology File Commands . . . 575
28.8. Trajectory File Commands . . . 580
28.9. Action Commands . . . 587
28.10Analysis Commands . . . 648
28.11Analysis Examples . . . 682

29. pytraj **683**
29.1. Introduction . . . 683
29.2. Development . . . 683
29.3. Documentation and examples . . . 683

30. MMPBSA.py **687**
30.1. Introduction . . . 687
30.2. Preparing for an MM/PB(GB)SA calculation . . . 688
30.3. Running MMPBSA.py . . . 690
30.4. Python API . . . 702

31. MM_PBSA **708**
31.1. General instructions . . . 708
31.2. Input explanations . . . 709

32. FEW **716**
32.1. Installation . . . 716
32.2. Overview of workflow steps and minimal input . . . 718
32.3. Common setup of molecular dynamics simulations . . . 718
32.4. Workflow for automated MM-PBSA & MM-GBSA calculations (WAMM) . . . 725

32.5. Linear interaction energy workflow (LIEW) . . . 732
32.6. Thermodynamic integration workflow (TIW) . . . 736

33. XtalAnalyze **744**
33.1. XtalAnalyze.sh . . . 744
33.2. XtalPlot.sh . . . 746
33.3. md2map.sh . . . 747

34. SAXS **749**
34.1. Introduction and theory . . . 749
34.2. Usage . . . 750

VI. NAB and AmberLite **753**

35. NAB: Introduction **755**
35.1. Background . . . 756
35.2. Methods for structure creation . . . 757
35.3. Compiling nab Programs . . . 759
35.4. Parallel Execution . . . 759
35.5. First Examples . . . 760
35.6. Molecules, Residues and Atoms . . . 763
35.7. Creating Molecules . . . 763
35.8. Residues and Residue Libraries . . . 764
35.9. Atom Names and Atom Expressions . . . 766
35.10Looping over atoms in molecules . . . 767
35.11Points, Transformations and Frames . . . 768
35.12Creating Watson Crick duplexes . . . 769

36. NAB: Language Reference **778**
36.1. Language Elements . . . 778
36.2. Higher-level constructs . . . 780
36.3. Statements . . . 786
36.4. Structures . . . 788
36.5. Functions . . . 789
36.6. Points and Vectors . . . 790
36.7. String Functions . . . 791
36.8. Math Functions . . . 791
36.9. System Functions . . . 792
36.10I/O Functions . . . 792
36.11Molecule Creation Functions . . . 795
36.12Creating Biopoloymers . . . 796
36.13Fiber Diffraction Duplexes in NAB . . . 796
36.14Reduced Representation DNA Modeling Functions . . . 797
36.15Molecule I/O Functions . . . 798
36.16Other Molecular Functions . . . 798
36.17Debugging Functions . . . 800
36.18Time and date routines . . . 801
36.19Computational resource consumption functions . . . 801

37. NAB: Rigid-Body Transformations **802**
37.1. Transformation Matrix Functions . . . 802
37.2. Frame Functions . . . 802

37.3. Functions for working with Atomic Coordinates . . . 803
37.4. Symmetry Functions . . . 803
37.5. Symmetry server programs . . . 805

38. NAB: Distance Geometry **809**
38.1. Metric Matrix Distance Geometry . . . 809
38.2. Creating and manipulating bounds, embedding structures . . . 810
38.3. Distance geometry templates . . . 814
38.4. Bounds databases . . . 816

39. NAB: Molecular mechanics and dynamics **818**
39.1. Basic molecular mechanics routines . . . 818
39.2. NetCDF read/write routines . . . 828
39.3. Typical calling sequences . . . 831
39.4. Second derivatives and normal modes . . . 832
39.5. Low-MODe (LMOD) optimization methods . . . 833
39.6. The Generalized Born with Hierarchical Charge Partitioning (GB-HCP) . . . 845

40. NAB: Sample programs **848**
40.1. Duplex Creation Functions . . . 848
40.2. nab and Distance Geometry . . . 849
40.3. Building Larger Structures . . . 857
40.4. Wrapping DNA Around a Path . . . 863
40.5. Other examples . . . 869

41. amberlite: Some AmberTools-Based Utilities **870**
41.1. Introduction . . . 870
41.2. Coordinates and Parameter-Topology Files . . . 871
41.3. *pytleap*: Creating Coordinates and Parameter-Topology Files . . . 873
41.4. Energy Checking Tool: *ffgbsa* . . . 876
41.5. Energy Minimizer: *minab* . . . 876
41.6. Molecular Dynamics "Lite": *mdnab* . . . 877
41.7. MM(GB)(PB)/SA Analysis Tool: *pymdpbsa* . . . 878
41.8. Examples and Test Cases . . . 885

Bibliography **894**

Index **940**

Part I.

Introduction and Installation

1. Introduction

Amber is the collective name for a suite of programs that allow users to carry out molecular dynamics simulations, particularly on biomolecules. None of the individual programs carries this name, but the various parts work reasonably well together, and provide a powerful framework for many common calculations.[1, 2] The term *Amber* is also used to refer to the empirical force fields that are implemented here.[3, 4] It should be recognized however, that the code and force field are separate: several other computer packages have implemented the *Amber* force fields, and other force fields can be implemented with the *Amber* programs. Further, the force fields are in the public domain, whereas the codes are distributed under a license agreement.

The Amber software suite is divided into two parts: AmberTools17, a collection of freely available programs mostly under the GPL license, and Amber16, which is centered around the *pmemd* simulation program, and which continues to be licensed as before, under a more restrictive license. Amber16 represents a significant change from the most recent previous version, Amber14. (We have moved to numbering releases by the last two digits of the calendar year, so there is no version 13 or 15.) Please see `http://ambermd.org` for an overview of the most important changes.

AmberTools is a set of programs for biomolecular simulation and analysis. They are designed to work well with each other, and with the "regular" Amber suite of programs. You can perform many simulation tasks with AmberTools, and you can do more extensive simulations with the combination of AmberTools and Amber itself. Most components of AmberTools are released under the GNU General Public License (GPL). A few components are in the public domain or have other open-source licenses. See the *README* file for more information.

Everyone should read (or at least skim) this chapter. Even if you are an experienced Amber user, there may be things you have missed, or new features, that will help. There are also tips and examples on the Amber Web pages at *http://ambermd.org*. Although Amber may appear dauntingly complex at first, it has become easier to use over the past few years, and overall is reasonably straightforward once you understand the basic architecture and option choices. In particular, we have worked hard on the tutorials to make them accessible to new users. Thousands of people have learned to use Amber; don't be easily discouraged.

If you want to learn more about basic biochemical simulation techniques, there are a variety of good books to consult, ranging from introductory descriptions,[5–7] to standard works on liquid state simulation methods,[8–10] to multi-author compilations that cover many important aspects of biomolecular modelling.[11–15] Looking for "paradigm" papers that report simulations similar to ones you may want to undertake is also generally a good idea. If you are new to this field, Chapter 14 provides a basic introduction to force fields, along with details of how the parameters are encoded in Amber files.

1.1. Information flow in Amber

Understanding where to begin in AmberTools is primarily a problem of managing the flow of information in this package — see Fig. 1.1. You first need to understand what information is needed by the simulation programs (*sander, pmemd, mdgx* or *nab*). You need to know where it comes from, and how it gets into the form that these programs require. This section is meant to orient the new user and is not a substitute for the individual program documentation.

Information that all the simulation programs need (see the circles in Fig. 1.1):

1. Cartesian coordinates for each atom in the system. These usually come from X-ray crystallography, NMR spectroscopy, or model-building. They should generally be in Protein Data Bank (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. Generally, editing of these files is needed, and the *pdb4amber* script can do some of this.

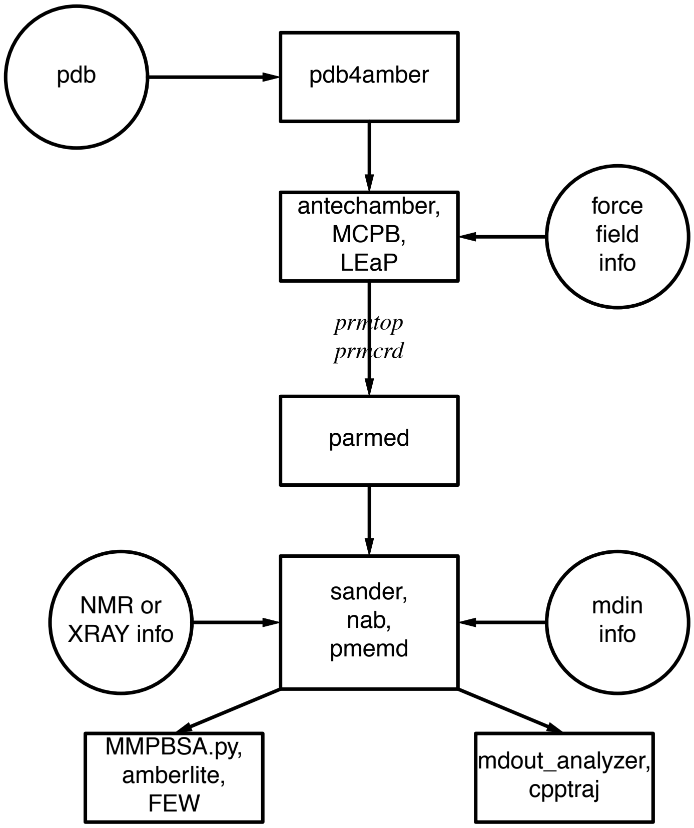


Figure 1.1.: *Basic information flow in Amber*

2. Topology: Connectivity, atom names, atom types, residue names, and charges. This information comes from the database, which is found in the *$AMBERHOME/dat/leap/lib* directory, 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, RNA, and common sugars and lipids. Topology information for other molecules (not found in the standard database) is kept in user-generated "residue files", which are generally created using *antechamber*.

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

4. Once the topology and coordinate files (often called *prmtop* and *prmcrd*, but any legal file names can be used) are created, the *parmed* script can be used to examine and verify these, and to make modifications. In particular, the *checkValidity* action will flag many potential problems.

5. Commands: The user specifies the procedural options and state parameters desired. These are specified in input files (named *mdin* by default) or in "driver" programs written in the NAB language.

1.1.1. Preparatory programs

LEaP is the primary program to create a new system in Amber, or to modify existing systems. It is available as the command-line program *tleap* or the GUI *xleap*. It combines the functionality of *prep*, *link*, *edit* and *parm* from much earlier versions of Amber.

pdb4amber generally helps in preparing pdb-format files coming from other places (such as *rcsb.org*) to be compatible with LEaP.

parmed provides a simple way to extract information about the parameters defined in a parameter-topology file. It can also be used to check that the parameter-topology file is valid for complex systems (see the *checkValidity* command), and it can also make simple modifications to this file very quickly.

antechamber is the main program to develop force fields for drug-like molecules or modified amino acids using the general Amber force field (GAFF). These can be used directly in LEaP, or can serve as a starting point for further parameter development.

MCPB.py provides a means to build, prototype, and validate MM models of metalloproteins and organometallic compounds. It uses the bonded plus electrostatics model to expand existing pairwise additive force fields. It is a reimplementation of MCPB in Python, with a more efficient workflow and many modeling processes from previous versions incorporated automatically.

IPMach.py provides a tool to facilitate the parameterization of nonbonded models (12-6 LJ model and 12-6-4 LJ-type model) for ions.

paramfit allows the generation of bonded force field parameters for any molecule by fitting to quantum data.

1.1.2. Simulation programs

sander (part of AmberTools) is the basic energy minimizer and molecular dynamics program. This program relaxes the structure by iteratively moving the atoms down the energy gradient until a sufficiently low average gradient is obtained. The molecular dynamics portion generates configurations of the system by integrating Newtonian equations of motion. MD will sample more configurational space than minimization, and will allow the structure to cross over small potential energy barriers. Configurations may be saved at regular intervals during the simulation for later analysis, and basic free energy calculations using thermodynamic integration may be performed. More elaborate conformational searching and modeling MD studies can also be carried out using the *sander* module. This allows a variety of constraints to be added to the basic force field, and has been designed especially for the types of calculations involved in NMR, Xray or cryo-EM structure refinement.

pmemd (part of Amber) is a version of *sander* that is optimized for speed and for parallel scaling; the *pmemd.cuda* variant runs on GPUs. The name stands for "Particle Mesh Ewald Molecular Dynamics," but this code can now also carry out generalized Born simulations. The input and output have only a few changes from *sander*.

mdgx is a molecular dynamics engine with functionality that mimics some of the features in *sander* and *pmemd*, but featuring simple C code and an atom sorting routine that simplifies the flow of information during force calculations. The principal purpose of *mdgx* is to provide a tool for redesign of the basic molecular dynamics algorithms and models, and for supporting new models for parameter development.

NAB (Nucleic Acid Builder) is a language that can be used to write programs to perform non-periodic simulations, most often using an implicit solvent force field.

1.1.3. Analysis programs

mdout_analyzer.py is a simple-to-run Python script that will provide summaries of information that is in the output files from *sander* or *pmemd*.

cpptraj is the main trajectory analysis utility (written in C++) for carrying out superpositions, extractions of coordinates, calculation of bond/angle/dihedral values, atomic positional fluctuations, correlation functions, analysis of hydrogen bonds, etc. See Chap. 28 for more information.

pytraj is a Python wrapper for cpptraj. It is written to introduce more flexibility in data analysis by combining with Python's rich echosystems (such as numpy, scipy, ipython-notebook ...).

pbsa is an analysis program for solvent-mediated energetics of biomolecules. The *pbsa.cuda* variant runs on GPUs. It can be used to perform both electrostatic and non-electrostatic continuum solvation calculations with input coordinate files from molecular dynamics simulations and other sources (in the pqr format). It also supports visualization of solvent-mediated electrostatic potentials in various visualization programs. See Chap. 6 for more information.

MMPBSA.py is a python script that automates energy analysis of snapshots from a molecular dynamics simulation using ideas generated from continuum solvent models. (There is also an older perl script, called *mm_pbsa.pl*, that has similar functionality.)

FEW (Free energy workflow) automates free energy calculations of protein-ligand binding using TI, MM/PBSA-type, or LIE calculations.

amberlite is small set of NAB programs and python scripts that implement a limited set of MD simulations and mm-pbsa (or mm-gbsa) analysis, aimed primarily at the analysis of protein-ligand interactions. These tools can be useful in their own right, or as a good introduction to Amber and a starting point for more complex calculations.

XtalAnalyze a set of utilities for analyzing crystal simulation trajectories. See Chapter 33 for more information.

1.2. List of programs

Amber is comprised of a large number of programs designed to aid you in your computational studies of chemical systems, and the number of released tools grows regularly. This section provides a list of the main programs included with AmberTools. Each program included in the suite is listed here with a very brief description of its main function along with which chapter in the manual a more thorough description can be found. Note: there are some additional programs that are part of the MTK++ suite that are not listed here; see that documentation for more information.

AddToBox A program for adding solvent molecules to a crystal cell. See Subsection 16.3.

CartHess2FC.py A program to derive the force constants based on Cartesian Hessian matrix using Seminario method. See Subsection 15.8.2.5.

car_to_files.py A program program to generate the mol2 and PDB files based on the car file. SeeSubsection 15.8.2.8.

ChBox A program for changing the box dimensions of an Amber restart file. See Subsection 16.4.

CheckMD A program for automated checking of an MD simulation. Run the program without options for usage statement.

IPMach.py A python program for facilitating the parameterization of the nonbonded models of ions. See Subsection 15.8.2.2.

MCPB.py A python version of MCPB with optimized workflow. See Subsection 15.8.2.1.

MMPBSA.py A program to post-process trajectories to calculate binding free energies according to the MM/PBSA approximation. See Chapter 30.

MTKppConstants Lists the constants used in MTK++. Run the program without arguments to get the full list.

OptC4.py optimizes the C4 terms in the metal-site-complex of a protein system. See Subsection 15.8.2.4.

PdbSearcher.py a python version of Pdbsearcher, a program in MTK++. See Subsection 15.8.2.3.

PropPDB A program for propagating a PDB structure. See Subsection 16.2

ProScrs.py A program for cutting and capping the protein segment into clusters. SeeSubsection 15.8.2.7.

UnitCell A program for recreating a crystallographic unit cell from a PDB structure. See Subsection 16.1

acdoctor A tool to diagnose what may be causing antechamber to fail. See Subsection 15.5.1

am1bcc A program called by antechamber to calculate AM1-BCC charges during ligand parametrization. It can be used as a standalone program, with the options printed when you enter the program name with no arguments. See Section 15.4

ambpdb A program to convert an Amber system (prmtop and inpcrd/restart) into a PDB, MOL2, or PQR file. See Section 27.1

ante-MMPBSA.py A program to create the necessary, self-consistent prmtop files for *MMPBSA* with a single starting topology file. See Subsection 30.2.2

antechamber A program for parametrizing ligands and other small molecules. See Chapter 15

atomtype A program called by *antechamber* to judge the atom types in an input structure. It can be used as a standalone program. If you provide no arguments, it prints out the usage statement. See Section 15.4

bondtype A program called by *antechamber* to judge what types of bonds exist in a given input structure. It can be used as a standalone program. If you provide no arguments, it prints out the usage statement. See Section 15.4

charmmlipid2amber.py A script that converts a PDB created with the CHARMM-GUI lipid builder into one recognized by Amber and AmberTools programs. If you provide no arguments, it prints out the usage statement.

cpinutil.py A program to create a constant pH input (CPin) file from a PDB file. If you provide no arguments, you get the usage statement.

cpptraj A versatile program for trajectory post-processing and data analysis. See Chapter 28

cphstats A program that computes protonation state statistics from constant pH simulations. See Section 22.7

elsize A program that estimates the effective electrostatic size of a given input structure. See Section 4.2.1

espgen A program called by *antechamber* to generate ESP files during ligand or small molecule parametrization. If you provide no arguments, it prints out the usage statement.

espgen.py A python version of espgen. See Subsection 15.8.2.6.

ffgbsa A program that calculates MM/GBSA energies as part of the amberlite package.

FEW.pl A program to automate the workflow for free energy calculations. See Chapter 32

gbnsr6 A program to compute a surface-area-based Generalized Born solvation free energy. See Section 5

hcp_getpdb A program that adds necessary sections to a topology (prmtop) file so it can be used for the HCP GB approximation. See Section 39.6

makeANG_RST A program to create angle restraints for use with sander's nmropt=1 facility.

makeCHIR_RST A program to create chiral restraint file for use with sander's nmropt=1 facility

makeDIP_RST.cyana A program to make restraints based on dipole information from CYANA for use with sander's nmropt=1 facility

makeDIST_RST A program to make distance restraints for use with sander's nmropt=1 facility

matextract Part of the symmetry definition programs, used to print matrices dumped to stdin to stdout. See Subsection 37.5.5

matgen Generate symmetry-transformation matrices. Part of the symmetry definition programs. See Subsection 37.5.1

matmerge Merges symmetry-transformation matrices into one matrix transformation matrix. Part of the symmetry definition programs. See Subsection 37.5.3

matmul Multiplies matrices. Part of the symmetry definition programs. See Subsection 37.5.4

mdgx An explicit solvent, PME molecular dynamics engine. See Chapter 26

mdnab Implicit solvent MD program written in NAB as part of Amberlite (Chapter 41)

mdout_analyzer.py A script that allows you to rapidly analyze and graph data from sander/pmemd output files. See Section 27

minab Implicit solvent minimization program written in NAB as part of Amberlite (Chapter 41)

mm_pbsa.pl Older perl script for performing MM/PBSA calculations. New users are encouraged to use MMPBSA.py instead.

mm_pbsa_statistics.pl Complementary script to mm_pbsa.pl to compute MM/PBSA statistics from a completed mm_pbsa calculation.

mm_pbsa_nabnmode Program for performing minimizations and normal mode analyses on biomolecules through mm_pbsa.pl.

mmpbsa_py_energy A NAB program written to calculate energies for *MMPBSA* using either GB or PB solvent models. It can be used as a standalone program that mimics the *imin=5* functionality of *sander*, but it is called automatically inside *MMPBSA*. See *MMPBSA* mdin files as example input files for this program. Providing the *–help* or *-h* flags prints the usage message.

mmpbsa_py_nabnmode A NAB program written to calculate normal mode entropic contributions for *MMPBSA*. This can really only be used by *MMPBSA*.

molsurf A program that calculates a molecular surface area based on input PQR files and a probe radius. Providing no arguments prints the usage message.

nab Stands for Nucleic Acid Builder. NAB is really a compiler that provides a convenient molecular programming language loosely based on C. See Chapter 35 and other related chapters.

new_crd_to_dyn Sets up a Tinker-style .DYN file from a coordinate file. This is part of the Tinker-to-Amber conversion program set.

new_to_old_crd Converts new Tinker-style coordinate files to old Amber-style coordinates. Part of the Tinker-to-Amber conversion program set.

nmode An outdated program to compute normal modes for biomolecules. You are encouraged to use NAB instead. See Section 39.1

paramfit Improves force field parameters by fitting to quantum data. See Chapter 11

parmcal Calculates parameters for given angles and bonds interactively. See Subsection 15.5.2

parmchk2 A program that analyzes an input force field library file (mol2 or amber prep), and extracts relevant parameters into an frcmod file. See Subsection 15.1.2

parmed A program for querying and manipulating prmtop files. See Section 14.2

pbsa A program for computing electrostatic and non-electrostatic continuum solvation free energies. See Chapter 6

pbsa.cuda A GPU-accelerated version of *pbsa*. See Chapter 6

pdb4amber A program to prepares PDB files for use in *leap*. See Section 12.4

pmemd A performance- and parallel-optimized dynamics engine implementing a subset of sander's functionality

pmemd.cuda A GPU-accelerated version of pmemd

prepgen A program used as part of *antechamber* that generates an Amber prep file. Use no arguments to print the usage message. See Section 15.4

pymdpbsa A full analysis tool for MD(GB,PB)/SA computations. See Section 41.7

pytleap A user-friendly wrapper for simple *leap* and *antechamber* runs to prepare topology and coordinate files for Amber. See Section 41.3

pytraj A Python progam binding to cpptraj. See Section 29

reduce A program for adding or removing hydrogen atoms to a PDB. See Section 12.5

residuegen A program to automate the generation of an Amber residue template (i.e. Amber prep file). See Subsection 15.5.3

resp A program typically called by *antechamber* and R.E.D tools to perform a Restrained ElectroStatic Potential calculation for calculating partial atomic charges. Use no arguments to get the usage message

respgen A program called by *antechamber* to generate RESP input files. See Section 15.4

rism1d A 1D-RISM solver. See Section 7.4

rism3d.snglpnt A 3D-RISM solver for single point calculations. See Section 7.6

sander The main engine used for running molecular simulations with Amber. Originally an acronym standing for Simulated Annealing with Nmr-Derived Energy Restraints.

saxs_rism A program to compute small (wide) angle X-ray scattering curve from 3D-RISM output

saxs_md A program to compute small (wide) angle X-ray scattering curve from MD trajectories

softcore_setup.py A program to aid in softcore TI setup for *sander*. Use no arguments to get a usage message.

sqm Semiempirical (or Stand-alone) Quantum Mechanics solver. See Chapter 9

tinker_to_amber Converts a Tinker analout and parameter file into an Amber-compatible topology file.

tleap A script that calls teLeap with specific setup command-line arguments. See Chapter 13

transform Applies matrix transformations to a structure. Part of the symmetry definition programs. See Subsection 37.5.6

tss_init A program to do some matrix stuff. See Section 37.5

tss_main A program to do some matrix stuff. See Section 37.5

tss_next A program to do some matrix stuff. See Section 37.5

ucpp A program to do some source code preprocessing. You should never actually use this program—it is used by *nab*.

xaLeap A graphical program for creating Amber topology files. This program is called through the *xleap* script, so you should never actually invoke this program directly.

xleap A script that calls xaLeap with specific setup command-line arguments. See Chapter 13

xparmed A graphical front-end to ParmEd functionality (i.e., parameter file editing and querying). See Section 14.2

2. Installation

This chapter gives an overview of how to install and test your distribution. The Amber web page (`http://ambermd.org`) has additional instructions and hints for various common operating systems. Look for the "Running Amber on ...." links. Once you have downloaded the distribution files, do the following:

1. First, extract the files in some location (we use */home/myname* as an example here):

   ```
   cd /home/myname
   tar xvfj AmberTools17.tar.bz2  #  (Note: extracts in an
                                  #   "amber16" directory)
   tar xvfj Amber16.tar.bz2    # (only if you have licensed Amber 16!)
   ```

2. Next, set your AMBERHOME environment variable:

   ```
   export AMBERHOME=/home/myname/amber16     # (for bash, zsh, ksh, etc.)
   setenv AMBERHOME /home/myname/amber16     # (for csh, tcsh)
   ```

   Be sure to change the "/home/myname" above to whatever directory is appropriate for your machine, and be sure that you have write permissions in the directory tree you choose.

3. Next, you may need to install some compilers and other libraries. Details depend on what OS you have, and what is already installed. Package managers can greatly simplify this task. For example for Debian-based Linux systems (such as Ubuntu), the following command should get you what you need:

   ```
   sudo apt-get install csh flex gfortran g++ xorg-dev \
         zlib1g-dev libbz2-dev patch python-tk python-matplotlib
   ```

   See *http://ambermd.org/amber_install.html* for more information, and for requirements for other variants of Linux, and for Macintosh OSX.

4. Now, in the *AMBERHOME* directory, run the configure script:

   ```
   cd $AMBERHOME
   ./configure --help
   ```

   will show you the options. Choose the compiler and flags you want; for most systems, the following should work:

   ```
   ./configure gnu
   ```

   This step will also check to see if there are any updates and bug fixes that have not been applied to your installation, and will apply them (unless you ask it not to). If the configure step finds missing libraries, go back to Step 3. This step will also ask if you want to install a compatible Python executable for the Python programs in Amber (including MMPBSA.py, MCPB.py, ParmEd, pysander, pytraj, pdb4amber, and the rest of amberlite). Since Amber now requires Python 2.7 or later, along with numpy, scipy, and matplotlib to enable all of its functionality, configure now provides an option to download a compatible Python from Continuum IO (via *miniconda*) installed in the Amber directory for use with Amber programs. See Section 2.2 for more details. As long as your default Python has the required prerequisites installed, configure will simply select that Python for use with Amber.

   *Do **not** choose any parallel options at this step!* You will need to install the serial version first; options for parallel builds are described below at Step 8.

   To automatically accept all the suggestions from configuring process, you can pipe the output of"yes" to configure:

```
yes | ./configure gnu
```

5. The configure step will create two resource files in the AMBERHOME directory: *amber.sh* and *amber.csh*. These sourceable scripts will set up your shell environment correctly for Amber:

   ```
   source /home/myname/amber16/amber.sh  # for bash, zsh, ksh, etc.
   source /home/myname/amber16/amber.csh # for csh, tcsh
   ```

   Of course, `/home/myname/amber16` should be adjusted for your AMBERHOME. Adding these commands to your login resource file (e.g., ~/.bashrc, ~/.cshrc, ~/.zshrc, etc.) will set up your environment every time you start a new shell. Note, this step is absolutely necessary to run any of the Python modules included with Amber.

6. Then,

   ```
   make install
   ```

   will compile the codes. If this step fails, try to read the error messages carefully to identify the problem.

7. This can be followed by

   ```
   make test
   ```

   which will run tests and will report successes or failures.

   Where "`possible FAILURE`" messages are found, go to the indicated directory under `$AMBERHOME/AmberTools/test` or `$AMBERHOME/test`, and look at the "*.dif" files. Differences should involve round-off in the final digit printed, or occasional messages that differ from machine to machine (see below for details). As with compilation, if you have trouble with individual tests, you may wish to comment out certain lines in the Makefiles (i.e., `$AMBERHOME/AmberTools/test/Makefile` or `$AMBERHOME/test/Makefile`), and/or go directly to the test subdirectories to examine the inputs and outputs in detail. For convenience, all of the failure messages and differences are collected in the `$AMBERHOME/logs` directory; you can quickly see from these if there is anything more than round-off errors.

   The nature of molecular dynamics, is such that the course of the calculation is very dependent on the order of arithmetical operations and the machine arithmetic implementation, *i.e.*, the method used for round-off. 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 round-off behavior; (a + b) + c is not necessarily the same as a + (b + c). Different optimization levels will affect operation order, and may therefore affect the course of the calculations.

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

   Note: If you have untarred the `Amber16.tar.bz2` file, then steps 1-6 will install and test both *AmberTools* and *Amber*; otherwise it will just install and test *AmberTools.* If you license *Amber* later, just come back and repeat steps 1-6 again.

8. If you are new to Amber, you should look at the tutorials and this manual and become familiar with how things work. If and when you wish to compile parallel (MPI) versions of *Amber*, do this:

```
cd $AMBERHOME
./configure -mpi <....other options....> <compiler-choice>
make install
# Note the value below may depend on your MPI implementation
export DO_PARALLEL="mpirun -np 2"
make test
# Note, some tests, like the replica exchange tests, require more
# than 2 threads, so we suggest that you test with either 4 or 8
# threads as well
export DO_PARALLEL="mpirun -np 8"
make test
```

This assumes that you have installed MPI and that *mpicc* and *mpif90* are in your PATH. Some MPI installations are tuned to particular hardware (such as infiniband), and you should use those versions if you have such hardware. Most people can use standard versions of either *mpich* or *openmpi*. To install one of these, use one of the simple scripts that we have prepared:

```
cd $AMBERHOME/AmberTools/src
./configure_mpich <compiler-choice>    OR
./configure_openmpi <compiler-choice>
```

Follow the instructions of these scripts, then return to beginning of step 7.

Amber 16 brings with it an additional flag (*-intelmpi*) to enable use of the Intel® MPI Library. Use this instead of *-mpi* in step 8 and ensure that you have installed the Intel® MPI Library, and that *mpiicc* and *mpiifort* are in your PATH.

Some notes about the parallel programs in AmberTools:

1. The MPI version of *nab* is called *mpinab*, by analogy with *mpicc* or *mpif90*: *mpinab* is a compiler that will produce an MPI-enabled executable from source code written in the NAB language. Before compiling *mpinab*, be sure that you are familiar with the serial version of *nab* and that you really need a parallel version. If you have shared-memory nodes, the OpenMP version might be a better alternative. See Section 35.4 for more information. (Note that *mpinab* is primarily designed to write driver routines that call MPI versions of the energy functions; it is not set up to write your own, novel, parallel codes.)

2. The MPI version of *MMPBSA.py* is called *MMPBSA.py.MPI*, and requires the package *mpi4py* to run. If it is not present in your Python standard library already, it will be built along with *MMPBSA.py.MPI* and placed in the `$AMBERHOME` prefix. If you have problems with *MMPBSA.py.MPI*, see if you get the same problems with the serial version, *MMPBSA.py*, to see if it is an issue with the parallel version or *MMPBSA.py* in general. Because we do not make or maintain the *mpi4py* source code, *MMPBSA.py.MPI* will not be available on platforms on which *mpi4py* cannot be built.

3. NAB and Cpptraj can also be compiled using OpenMP:

   ```
   ./configure -openmp <....other options....> <compiler-choice>
   make openmp
   ```

   Note that the OpenMP version of NAB has the same name as the single-threaded version. See section 35.4 for information on running the OpenMP version of NAB and section 28.1.6.2 for information on running the OpenMP version of Cpptraj.

4. See Section 18.6.5 for information about installing the GPU-accelerated versions of *pmemd*.

5. See Section 6.6.4 for information about installing the GPU-accelerated version of *pbsa*.

2.1. Uninstalling and cleaning

All of Amber and AmberTools is contained within the `$AMBERHOME` directory. So deleting this directory will completely remove Amber from your computer. However, there are cases when you may wish to remove some of the files and programs that were created when you compiled Amber; this section describes the Makefile rules that are provided for those purposes.

make clean This command will remove all of the temporary object files that were created when you compiled Amber (these files usually reside in the same directory as the source code). This is necessary, for instance, when you intend to build a new variant of the Amber programs (such as building the OpenMP-parallelized programs or the MPI-parallelized programs). This is done automatically by the `configure` script. The only time this would really be necessary is if you were modifying the compiler flags by editing the `config.h` file by hand or setting the `AMBERBUILDFLAGS` environment variable. These are considered advanced options.

make distclean This command is a sledgehammer—it deletes all of the temporary object files, installed libraries, and programs, as well as several third-party libraries (but not things you might have installed separately, like MPI implementations via the `configure_openmpi` or `configure_mpich` scripts). The purpose of this command is to return the source code tree to a pristine state, as though you just extracted a fresh copy of the source code. If you plan on changing the compiler that you use to build Amber, you should run this command first.

Note that none of the above commands reverse any of the updates that you have already applied—they *only* impact files and programs that were created during the install process.

2.2. Python in Amber

Recent developments in Amber have increasingly added the Python programming language as the core language of several of its components. In addition to standalone programs like MMPBSA.py, MCPB.py, and ParmEd, a growing number of components also expose a substantial fraction of Amber functionality through Python APIs, like pysander, ParmEd, and pytraj.

Amber's Python programs and libraries take advantage of the rich, mature, and robust scientific computing ecosystem founded on the *numpy* numerical programming package. Standard dependencies for many Amber Python packages include *numpy*, *scipy*, *matplotlib*, and *tkinter*, just to name a few. As the Python language matures, modern tools begin to lose compatibility with older versions of Python. Many key components of Amber, beginning with version 16, are compatible only with versions of Python 2.7, 3.4 (and later).

Since some Linux variants ship with versions of Python older than this, AmberTools includes a *configure_python* script that will download and install a compliant Python version inside the Amber install directory in order to drive Amber's Python components. If the standard configure process does not detect a Python that is both new enough, and has enough core components installed to run the bulk of Amber Python capabilities, you will be asked if you want *configure* to download and install a version of Miniconda (shipped by Continuum IO) for you.

If you are not an experienced Python user, we recommend that you choose "yes" and allow AmberTools to create and manage the Python environment it needs for all of its tools. If you *are* an experienced Python user, and would like to make use of the various AmberTools Python APIs, see the options for installing AmberTools Python packages to the standard Python environments below.

This is a list of packages that AmberTools will install if user chose "yes": python2.7, numpy, scipy, cython, ipython, notebook, matplotlib. The total downloaded file size is ~100 MB and the total installed size will be ~ 0.5GB. User can access Python via *amber.python*.

2.2.1. Picking your Python interpreter

By default, *configure* will attempt to use the *python* executable in $AMBERHOME/miniconda/bin. If it is not found, you will be notified and asked if you want to install Miniconda. If so, AmberTools will download and install this version of Python 2.7 (and will always look for that version first in future *configure* calls unless told to look elsewhere).

You can use the `--with-python` flag to configure (via `--with-python /path/to/python`) in order to specify a particular Python executable you wish to use with Amber. This flag overrides the checks for existing prerquisites, and should only be used if you know what you are doing. In this case, you can install required programs by yourself:

```
# Via conda (recommended)
conda install --file AmberTools/src/python_requirement.txt
# Or via traditional"pip" package manager
pip install -r AmberTools/src/python_requirement.txt
```

2.2.2. Python package install location

By default, AmberTools attempts to install Python packages to `$AMBERHOME/lib/pythonX.Y`, where X.Y is the version of Python was found (or assigned) by *configure*. If *configure* installed a version of Miniconda, this is `$AMBERHOME/lib/python2.7`. The amber.sh and amber.csh resource scripts then add this path to your PYTHONPATH environment variable to ensure that the Python runtime can find these packages. In some cases, like with the standard Phenix bindings, external programs may depend on the Python packages from Amber being installed to that location, so choosing one of the other options risks breaking third-party extensions. For somebody experienced with using and developing Python, such problems will be easy to spot and fix. If you are unsure, you are suggested to keep the default behavior.

In many cases, however, users may have sophisticated Python environments set up (through *conda* or *virtualenv*, for instance) and would like Amber's Python components to mesh more naturally in their Python environment than simply being dumped to `$AMBERHOME/lib/pythonX.Y`. For these use cases, the `--python-install` option was added to *configure*, which takes either *user*, *local*, or *global* as arguments. *local* is the default behavior. Selecting *user* installs all Python packages via the command

```
python setup.py install --user
```

which makes the packages automatically visible to the Python runtime, but does not require modifying any system-level directories that may require root access (however, it also will not work for any other user accounts). The *global* option will simply run

```
python setup.py install
```

and try to install all of the Amber modules and packages to the root level Python *site-packages* directory. This will require root access if you are using the system Python (not recommended), but may not require any special permissions if using a local installation of Python, such as that from Anaconda or Miniconda.

2.2.3. Supported Python versions

Users are encouraged to use Python versions 2.7 and 3.4 (or greater) since those versions have been verified to work with all Python components of Amber (assuming other prerequisites, like numpy and/or scipy are met). Python 2.6 is known *not* to work with pytraj and ParmEd (and, by extension, any package that depends on ParmEd, like MMPBSA.py).

Different components of AmberTools support different versions of Python. Some codes (like pytraj, ParmEd and pdb4amber) work unchanged in both Python 2.7 and Python 3.x, while others need to be converted using *2to3* upon installation. If users want to combine AmberTools (such as pysander, ParmEd) with 3rd party packages like phenix, PyRosetta, users need to use Python 2.7 version since those packages have not yet supported Python 3.x.

2.2.4. Amber's Python environment and installing new components

If you said "yes" to allow configure to install Python from Miniconda, or you ran the configure_python script directly, Python is installed to $AMBERHOME/miniconda. Symbolic links to the main *python*, *ipython*, *jupyter* and *conda* executables are created in `$AMBERHOME/bin` and named *amber.python*, *amber.ipython*, *amber.jupyter*, and *amber.conda*, respectively. This is done to avoid potential collisions with other Python environments that may be present.

The *conda* program is the Python package manager that is used to install new Python packages. So if you need to install a new package, you can use the *amber.conda* program to do so. For example, to install the *pandas* library, use the command

```
amber.conda install pandas
```

You are referred to the Continuum IO documentation about Anaconda (and Miniconda) for further information. See https://www.continuum.io/documentation for more information.

2.3. Applying Updates

For most users, simply running the configure script and responding 'yes' to the update request will automatically download and apply all patches. This section describes the main updating script responsible for managing updates. We suggest that you at least skim the first section on the basic usage—particularly the note about the *`--version`* flag for if/when you ask for help on the mailing list.

2.3.1. Basic Usage

Updates to AmberTools and Amber are downloaded, applied, and managed automatically using the Python script *update_amber*. This script works on every version of Python from Python 2.4 through the latest Python 3 release. The *configure* script in `$AMBERHOME` automatically uses *update_amber* to search for available updates to AmberTools (and Amber when present) unless explicitly disabled with the *`--no-updates`* flag (which must be the first option to *configure*). If any are available, you will be asked if you want them downloaded and applied. This script resides in `$AMBERHOME` and can be executed from anywhere (it will verify that `AMBERHOME` is set properly), but if moved from `AMBERHOME`, it will not work. There are 3 main operating modes, or actions, that you can perform with them:

- *`$AMBERHOME/update_amber --check-updates`* : This option will query the Amber website for any updates that have been posted that have not been applied to your installation. If you think you have found a bug, this is helpful to try first before emailing with problems since your bug may have already been fixed.
- *`$AMBERHOME/update_amber --version`* : This option will return which patches have been applied to the current tree so far. When emailing the Amber list with problems, it is important to have the output of this command, since that lets us know exactly which updates have been applied.
- *`$AMBERHOME/update_amber --update`* : This option will go to the Amber website, download all updates that have not been applied to your installation, and apply them to the source code. ***`Note that you will have to recompile any affected code for the changes to take effect!`***

2.3.2. Advanced options

update_amber has additional functionality as well that allows more intimate control over the patching process. For a full list of options, use the *`--full-help`* command-line option. These are considered advanced options.

- *`$AMBERHOME/update_amber --download-patches`* : Only download patches, do not apply them
- *`$AMBERHOME/update_amber --apply-patch=<PATCH>`* : This will apply a third-party patch
- *`$AMBERHOME/update_amber --reverse-patch=<PATCH>`* : Reverses a third-party patch file that was applied via the *`--apply-patch`* option (see above).
- *`$AMBERHOME/update_amber --show-applied-patches`* : Shows details about each patch that has been applied (including third-party patches)

- `$AMBERHOME/update_amber --show-unapplied-patches` : Shows details about each patch that has been downloaded but not yet applied.
- `$AMBERHOME/update_amber --remove-unapplied` : Deletes all patches that have been downloaded but not applied. This will force *update_amber* to download a fresh copy of that patch.
- `$AMBERHOME/update_amber --update-to AmberTools/#,Amber/#` : This command will apply all patches necessary to bring AmberTools up to a specific version and Amber up to a specific version. Note, no updates will ever be reversed using this command. You may specify only an AmberTools version or an Amber version (or both, comma-delimited). No patches are applied to an omitted branch.
- `$AMBERHOME/update_amber --revert-to AmberTools/#,Amber/#` : This command does the same as `--update-to` described above, except it will only reverse patches, never apply them.

update_amber will also provide varying amounts of information about each patch based on the verbosity setting. The verbose level can be set with the `--verbose` flag and can be any integer between 0 and 4, inclusive. The default verbosity level changes based on how many updates must be described. If only a small number of updates need be described, all details are printed out. The more updates that must be described, the less information is printed. If you manually set a value on the command-line, it will override the default. These values are described below (each level prints all information from the levels before plus additional information):

- 0: Print out only the name of the update file (no other information)
- 1: Also prints out the name of the program(s) that are affected
- 2: Also prints out the description of the update written by the author of that update.
- 3: Also prints the name of the person that authored the patch and the date it was created.
- 4: Also prints out the name of every file that is modified by the patch.

2.3.3. Internet Connection Settings

If update_amber ever needs to connect to the internet, it will check to see if http://ambermd.org can be contacted within 10 seconds. If not, it will report an error and quit. If your connection speed is particularly slow, you can lengthen this timeout via the `--timeout` command-line flag (where the time is given in seconds).

Proxies By default, *update_amber* will attempt to contact the internet through the same mechanism as programs like *wget* and *curl*. For users that connect to the internet through a proxy server, you can either set the `http_proxy` environment variable yourself (in which case you can ignore the rest of the advice about proxies here), or you can configure *update_amber* to connect to the internet through a proxy. To set up update_amber to connect to the internet through a proxy, use the following command:

```
$AMBERHOME/update_amber --proxy=<PROXY_ADDRESS>
```

You can often find your proxy address from your IT department or the preferences in your favorite (configured) web browser that you use to surf the web. If your proxy is authenticated, you will also need to set up a user:

```
$AMBERHOME/update_amber --proxy-user=<USERNAME>
```

If you have set up a user name to connect to your proxy, then you will be asked for your proxy password the first time *update_amber* attempts to utilize an online resource. (For security, your password is never stored, and will need to be retyped every time *update_amber* runs).

You can clear all proxy information using the `--delete-proxy` command-line flag—this is really only necessary if you no longer need to connect through any proxy, since each time you configure a particular proxy user or server it overwrites whatever was set before.

Mirrors If you would like to download Amber patches from another website or even a folder on a local filesystem, you can use the `--amber-updates` and `--ambertools-updates` command-line flags to specify a particular web address (must start with http://) or a local folder (use an absolute path). You can use the `--reset-remotes` command-line flag to erase these settings and return to the default Amber locations on http://ambermd.org.

If you set up online mirrors and never plan on connecting directly to http://ambermd.org, you can change the web address that *update_amber* attempts to connect to when it verifies an internet connection using the `--internet-check` command-line option.

2.4. Contacting the developers

Please send suggestions and questions to amber@ambermd.org. You need to be subscribed to post there; to subscribe, go to *http://lists.ambermd.org/mailman/listinfo/amber*. You can unsubscribe from this mailing list on the same site.

Part II.

Amber force fields

3. Molecular mechanics force fields

Amber is designed to work with several simple types of force fields, although it is most commonly used with parametrizations developed by Peter Kollman and his co-workers and "descendents". The traditional parametrization uses fixed partial charges, centered on atoms. Less commonly used modifications add polarizable dipoles to atoms, so that the charge description depends upon the environment; such potentials are called "polarizable" or "non-additive". An alternative is to use force fields originally developed for the CHARMM or tinker (AMOEBA) codes; these require a different setup procedure, which is described in Sections 14.2.2.8 (for CHARMM) and Chapter **??** (for AMOEBA). Chapter 14 provides a basic introduction to force fields, along with details of how the parameters are encoded in Amber files.

In previous versions of *AmberTools*, we including "combined" leaprc files (such as *leaprc.ff14SB*) that loaded, protein, nucleic acid and water models that worked well together. This was convenient for most users, but tended to obfuscate the important issue of deciding which force fields to use. Since various choices make good sense, and **we have implemented a new scheme** for users to specify the force fields they wish to use. Depending on what components are in your system, you may need to specify:

- a protein force field (recommended choice is *ff14SB*)
- a DNA force field (recommended choice is *OL15*)
- an RNA force field (recommended choice is *OL3*)
- a carbohydrate force field (recommended choice is *GLYCAM_06j*)
- a lipid force field (recommended choice is *lipid17*)
- a water model with associated atomic ions (more variable, but the most common choice is still *tip3p*); other popular choices are *spc/e*, *tip4pew*, and *OPC*. Not needed if you are using an implicit solvent model.
- a general force field, for organic molecules like ligands (recommended choice is *gaff2*)
- other components (such as modified amino acids or nucleotides, other ions), as needed

Notes:

1. You have to be careful if you try to adopt a "mix and match" strategy for different components. The recommended choices are designed to work well together, and have been fairly extensively tested. Use of other combinations requires a deeper knowledge of the nature and origin of force fields; see below and consult the original papers for more information. If you wish to combine proteins with nucleic acids, only the recommended combination above (or one where *leaprc.DNA.OL15* is replaced with *leaprc.DNA.bsc1*) is allowed.

2. In general, your input file to leap will begin with several commands to source the relevant leaprc files. For example the following preamble would allow you to include proteins, DNA, lipids, general components, water, and atomic ions like Na+ or Cl-, using the current recommended force fields:

```
source leaprc.protein.ff14SB
source leaprc.DNA.OL15
source leaprc.lipid17
source leaprc.water.tip3p
source leaprc.gaff2
```

Note that explicit solvent simulations now require you to load a `leaprc.water.xxxx` file; this is a change from AmberTools15 and earlier versions, where the TIP3P water model was loaded by default.

3. There are some leaprc files for older force fields in the *$AMBERHOME/dat/leap/cmd/oldff* directory. We no longer recommend these combinations, but we recognize that there may be reasons to use them, especially for comparisons to older simulations. See Section 3.9 for more information.
4. In particular, the *leaprc.ff14SB* file, in the oldff/ directory, is identical to the file of the same name in AmberTools15. In spite of its name, it is a "combined" file, with protein, DNA, RNA and water elements. This file might be of particular interest if you want to make sure that systems created the "new" way (with the leaprc files outlined above) are consistent with those using the older, "combined" method.

3.1. Proteins

In addition to the recommended file, *leaprc.protein.ff14SB*, there are a variety of alternatives for proteins; these are described in the following sections.

3.1.1. The ff14SB force field

```
leaprc.protein.ff14SB       This will load the files listed below
parm10.dat          ff10 force field parameters
frcmod.ff14SB       ff14SB modifications to parm10.dat
amino12.lib         topologies and charges for amino acids
amino12nt.lib       same, for N-terminal amino acids
amino12ct.lib       same, for C-terminal amino acids

leaprc.protein.ff14SBonly   This is the same as leaprc.protein.ff14SB, but will additionally load:
frcmod.ff99SB14         ff99SB backbone parameters with ff14SB atom types
```

ff14SB [16]is a continuing evolution of the *ff99SB* force field, primarily developed in the Simmerling group at Stony Brook University.[17] Several groups had noticed that the older *ff94* and *ff99* parameter sets did not provide a good energy balance between helical and extended regions of peptide and protein backbones. Another problem is that many of the *ff94* variants had incorrect treatment of glycine backbone parameters. *ff99SB* improved this behavior, presenting a careful reparametrization of the backbone torsion terms in *ff99* and achieves much better balance of four basic secondary structure elements (PP II, β, α_L, and α_R). A detailed explanation of the parametrization as well as an extensive comparison with many other variants of fixed-charge Amber force fields is given in the reference above. Briefly, dihedral term parameters were obtained through fitting the energies of multiple conformations of glycine and alanine tetrapeptides to high-level *ab initio* QM calculations. We have shown that this force field provides much improved proportions of helical versus extended structures. In addition, it corrected the glycine sampling and should also perform well for β-turn structures, two things which were especially problematic with most previous Amber force field variants.

Since 2006, a number of limitations of the *ff99SB* parameter sets became evident, and a new round of parameter optimization was undertaken. The changes mainly involve torsional parameters for the backbone and side chains. For **backbones**, experimental scalar coupling data for small solvated peptides became available [18] against which *ff99SB* was compared.[19] As *ff99SB* backbone dihedrals were fit based on gas-phase quantum data, we felt that slight empirical adjustments were worth pursuing. This was done to improve agreement with scalar coupling data, and we observed that this also improved stabilities of helical peptides.

The **side chain dihedral parameters** of *ff99SB* were the same as those of *ff99*. Residues such as isoleucine, leucine, aspartate, and asparagine (*cf. ff99SB-ILDN*) sample conformations different from those indicated by experiments. We therefore calibrated the dihedral corrections of the amino acid side chains against ab initio quantum mechanical energy surfaces. As *ff14SB* is an additive model, a key objective was to minimize dependence of *ff14SB* side chain parameters on particular backbone conformations. Therefore, side chain corrections were raised against potential energy surfaces including multiple backbone conformations. Moreover, the method of generating fitting data was adjusted to minimize backbone-dependence, including restraint of all backbone dihedrals and re-optimization of bonds and angles with each model. Whereas *ff12SB* also included restraints on all side chain dihedrals, restraint of one dihedral per side chain bond in *ff14SB* was found to further reduce backbone-dependence.

Together with new corrections for the backbone and the four amino acids addressed in *ff99SB-ILDN*, this work offers updated side chain dihedral corrections for lysine, arginine, glutamate, glutamine, methionine, serine, threonine, valine, tryptophan, cysteine, phenylalanine, tyrosine, and histidine. *ff14SB* enhances reproduction of experimentally indicated geometries over *ff99SB*. See Ref. [16] for futher details.

ff14SBonlysc, where *sc* stands for side chains, includes *ff99SB* backbone parameters with updated side chain parameters that were derived from ab initio quantum mechanics calculations (as were the *ff99SB* backbone corrections). This model is slightly different from *ff14SB,* which includes the *ff14SBonlysc* parameters as well as a small empirical correction to backbone parameters that was designed to improve agreement between NMR data and simulations in TIP3P water for short peptides. We are currently exploring whether this empirical correction also improves simulations in other water models, such as the *GBneck2* (igb=8) model. [20] Currently, it appears that igb=8 may work best with the fully quantum mechanics-based dihedral parameters included in *ff14SBonlysc*. Simulations performed in explicit water most likely benefit from the empirical corrections included in *ff14SB*.

3.1.2. The ff15ipq protein force field

```
leaprc.protein.ff15ipq        This will load the files listed below
parm15ipq_10.3.dat           force field parameters
amino15ipq_10.0.lib          topologies and charges for amino acids
aminont15_ipq10.0.lib        same, for N-terminal amino acids
aminoct15ipq_10.0.lib        same, for C-terminal amino acids
```

ff15ipq [21] continues the development begun with the *ff14ipq* force field [22, 23], but offers new, we hope better, parameter choices, data fitting, and validation. The physical assumptions behind the model are the same, but problems with *ff14ipq*, most generally the "stickiness" of polar groups in simulations, led to sweeping parameter changes. The pair-specific Lennard-Jones terms in *ff14ipq* were the problem, introducing an imbalance of protein:water and protein:protein interactions. They have been replaced by modified polar hydrogen radii and a consistent Lorentz-Berthlot combining rule as found in other Amber force fields. As a consequence, the entire charge set has changed, albeit slightly, and the the torsion parameters have been expanded and rederived. To further improve the internal potential energy surface, refitted angle parameters are included for the protein backbone. The new version comprises nearly 1,200 unique parameters, and *ff14ipq* is archived (use `oldff/leaprc.ff14ipq`) for backwards compatibility and comparisons.

The extended IPolQ charge derivation anticipates a workflow in which the final model must have charges roughly consistent with the polarization molecules experience in water, but also new torsion parameters which are often derived with quantum calculations of the system in vacuum. In the extended methodology, two sets of charges are fitted: one for the systems in vacuum, the other for systems in the condensed phase. The original IPolQ method [22] derives the appropriate condensed phase charges by fitting to the average electrostatic potential of polarized and unpolarized molecules, a process that harkens to linear response theory and implicitly accounts for the energetic cost of polarizing the system away from is gas phase equilibrium. The extended scheme draws on the vacuum phase electrostatic data a second time to make an alternative set of charges appropriate to describe the vacuum potential energy surface–the IPolQ charges themselves are, in fact, re-expressed as a perturbation of this gas phase charge set. Both sets of charges are derived in the same linear least squares fitting problem, with restraint equations weakly coupling the corresponding charges together. This creates charge sets for each phase related by a minimal perturbation, which can be assumed to be the effective, average polarization of the molecules when they enter solution. The charge set appropriate to the vacuum phase is then used when fitting torsion potentials to vacuum phase quantum mechanical energies, and the torsion potentials are transferred directly for use with the condensed-phase charge set in actual simulations, following the earlier assumption that the effective polarization of the molecules, and thereby any energetic consequences of entering the condensed phase, are captured in the charge perturbation.

All parameter optimization in *ff15ipq*, like its predecessor *ff14ipq*, is iterative: a generational learning scheme whereby the results of previous simulations and force field manipulations are submitted to quantum single point energy calculations and then added to the training data. As with *ff14ipq*, charges and gas-phase conformational energies are all taken at the MP2/cc-pVTZ level; *ff15ipq* takes the *ff14ipq* conformational energies as its starting point and expands the space nearly four-fold. We find that this crude form of machine learning is a good substitute for human intervention. As with *ff14ipq*, the iterative process led to an evolution in simulation performance over a variety of systems. We utilized these benchmarks to determine when the parameter set was ready for general release.

The new *ff15ipq* model [21] was derived with the SPC/E-b water model of Takemura and Kitao [24]. Returning to three-point water models improves performance of most Amber protein simulations on GPUs by about 30% due to the reduction in the overall number of particles; a smaller improvement can be seen on CPUs. While SPC/E-b is the recommended water model, the solvent reaction field potential observed in our IPolQ studies is consistent across three- and even some four-point waters: combinations of *ff15ipq* with TIP3P, the original SPC/E, and other water models are reasonable to try. One issue that may arise in some circumstances is the compatibility of the water model with ion parameters: we have set *ff15ipq* to reference ion parameters appropriate for the nearest water model available, SPC/E. However, for highly charged or dense ionic solutions this combination may be sub-optimal. With respect to compatibility with other macromolecular force fields such as sugars, lipids, or nucleic acids, we note that while the charge set is novel, the MP2/cc-pVTZ solution-phase IPolQ charges [22] are in fact quite similar to the Cornell charges derived at the HF/6-31G* level [25]. This result may support the long lifespan of that charge set, and makes it likely that *ff15ipq* will be compatible with other force fields designed at the common HF/6-31G* level.

ff15ipq has been validated on a larger number of test systems than its predecessor, and for much longer timescales. Multiple alpha-helical and beta-sheet peptides have been tested at a variety of temperatures, and numerous small proteins (the largest including lysozyme and the p53/MDM2 complex) have been simulated for timescales ranging from 4 to 10 microseconds, displaying excellent stability and also instability in cases where loops of the proteins or isolated peptides are known to be disordered. Various teething problems in the *ff14ipq* force field were solved by improvements to the data set or the fitting protocol itself, so we are increasingly confident that *ff15ipq* and future products of the IPolQ workflow will be reliable straight out of the automated parameter development phase. The entire data set and *mdgx* input file for deriving the torsion and angle parameters of *ff15ipq* will be released as supporting information in the upcoming publication on the force field. In the future we hope to build on the lineage of ff-ipq protein models to include other important areas of biological chemistry.

3.1.3. The fb15 ("force balance") protein force field

```
leaprc.protein.fb15       This will load the files listed below
frcmod.fb15             force field parameters
frcmod.tip3pfb          parameters for the force balance 3-point model
all_aminofb15.lib       topologies and charges for amino acids
all_aminontfb15.lib       same, for N-terminal amino acids
all_aminoctfb15.lib       same, for C-terminal amino acids
```

The files can be used for protein-water simulations using the "force-balance" approach described in Ref. [26, 27]. There is also a 4-point water model available, as described in section 3.5. For alkali and halide ions, the Joung-Cheatham parameters for TIP3P (or TIP4PEW) are recommended; see Section 3.6.

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

```
leaprc.protein.ff03.r1    loads the following files:
frcmod.ff03         For proteins: changes to parm99.dat, primarily in the
                    phi and psi torsions.
all_amino03.in      Charges and atom types for proteins
all_aminont03.in    For N-terminal amino acids
all_aminoct03.in    For C-terminal amino acids
```

The **ff03** force field [28, 29] is a modified version of *ff99* (described below). The main changes are that charges are now derived from quantum calculations that use a continuum dielectric to mimic solvent polarization, and that the ϕ and ψ backbone torsions for proteins are modified, with the effect of decreasing the preference for helical configurations. The changes are just for proteins; nucleic acid parameters are the same as in *ff99*.

The original model used the old (*ff94*) charge scheme for N- and C-terminal amino acids. This was what was distributed with Amber 9, and can still be activated by using *oldff/leaprc.ff03*. More recently, new libraries for the terminal amino acids have been constructed, using the same charge scheme as for the rest of the force field. This newer version (which is recommended for all new simulations) is accessed by using *leaprc.protein.ff03.r1*.

3.1.5. The Yang et al. (2003) united-atom force field

`frcmod.ff03ua`	`For proteins: changes to parm99.dat, primarily in the introduction of new united-atom carbon types and new side chain torsions.`
`uni_amino03.in`	`Amino acid input for building database`
`uni_aminont03.in`	`NH3+ amino acid input for building database.`
`uni_aminoct03.in`	`COO- amino acid input for building database.`

The **ff03ua** force field [30] is the united-atom counterpart of *ff03*. This force field uses the same charging scheme as *ff03*. In this force field, the aliphatic hydrogen atoms on all amino acid side-chains are united to their corresponding carbon atoms. The aliphatic hydrogen atoms on all alpha carbon atoms are still represented explicitly to minimize the impact of the united-atom approximation on protein backbone conformations. In addition, aromatic hydrogens are also explicitly represented. Van der Waals parameters of the united carbon atoms are refitted based on solvation free energy calculations. Due to the use of an all-atom protein backbone, the ϕ and ψ backbone torsions from *ff03* are left unchanged. The sidechain torsions involving united carbon atoms are all refitted. In this parameter set, nucleic acid parameters are still in all atom and kept the same as in *ff99*.

3.2. Nucleic acids

As with proteins, many features of the current force fields, including partial atomic charges, Lennard-Jones parameters, and most bond and angle terms, date back to force fields developed in the 1990's, and overviews of this work are available.[31, 32] The next breakthroughs in the Amber nucleic acid force field development came from observations from relatively longer simulations on the 50-100 ns time scale in the early 2000's.[33, 34] These simulations found systematic over-population of $\gamma =$ *trans* backbone geometries in nucleic acids. High level QM calculations were performed on models of sugars and phosphates, specifically a sugar-phosphate model[35] and a sugar-phosphate-sugar model,[36] which ultimately led to the *ff99-bsc0* parameterization.[35] For simulation of canonical DNA and RNA structures, the *ff99-bsc0* parameterization has proven rather successful. For non-canonical structures, particular those with loops or bulges, or χ flips, some anomalies have been noted.

3.2.1. RNA

With RNA, incorrect loop geometries, backbone sub-state populations and sugar pucker populations were observed in longer simulations. In addition to not being able to always maintain south puckers where found in RNA structures, multiple groups noticed a tendency for the RNA backbone to shift, putting χ into the high-*anti* region which leads to an opening of the duplex structure into a ladder-like configuration. Again, QM methods at various levels were employed to improve the χ distribution using relevant model systems. The most tested χ modifications are the "OL" modifications used in *ff14SB*.[37, 38] An alternative available with Amber is the Yildirim χ modifications (and also related modifications called TOR which alter ε/ζ as well)[39–41], and a systematic assessment and validation of these newer χ modifications is underway on a large series of RNA tetraloop structures. Note that small changes to a particular dihedral may lead to alteration in properties of related dihedrals, and may have unintended consequences. For example, the *ff99-bsc0* modifications tend to lock RNA sugar puckers mainly in the north, even with nucleotides in particular sequence contexts that prefer southern conformations. Moreover, the χ modifications tend to further destabilize $\gamma =$ trans. This suggests that to reliably improve the nucleic acid dihedrals, a more systematic approach across many dihedrals with simultaneous fitting may be more appropriate. Moreover,

Desired Behavior	Source these files	Notes
RNA		
ff99OL3	*leaprc.RNA.OL3*	*parmbsc0 α/γ [35] + χOL3 [38] to ff99*
ff99χ + bsc0	leaprc.RNA.YIL	parmbsc0 α/γ *[35]*+ Yildirim [39]χ mods to ff99.
ff99bsc0	oldff/leaprc.ff99bsc0	Contains parmbsc0 α/γ mods[35] to ff99.
“Rochester” torsions	leaprc.RNA.ROC	[43]
Modified nucleotides	leaprc.modrna08	parameters for modified nucleosides [44]
DNA		
OL15	*leaprc.DNA.OL15*	*Contains ε/ζOL1[45], χOL4[42] and βOL1[46] mods to ff14.*
bsc1	leaprc.DNA.bsc1	Contains parmbsc1 mods [47] to parmbsc0.

Table 3.1.: *How to specify nucelic acid force fields in LEaP; recommended variants are in italics.*

we no longer fully support the idea that parameters are transferable between DNA and RNA, or between purines and pyrimidines. For example, the *ff99-OL* modifications (with or without *ff99-bsc0*) improve the modeling of RNA, but lead to issues with DNA, most notably with quadruplex structures. Therefore recent work has focused on separate χ modifications for DNA.[42]

An alternative set of torsions for RNA, fit to quantum calculations has recently been reported by the Rochester group,[43] and can be loaded with the `leaprc.RNA.ROC` file.

3.2.2. DNA

A new set of parameters for the ε/ζ dihedral[45] and for the β dihedral[46] torsion for DNA have been developed using QM methods that include the solvation effects implicitly. This set of parameters have been tested with several double-stranded DNA systems including the Dickerson-Drew dodecamer, A-tracs, CG-rich duplexes, Z-DNA and G-quadruplexes. These modifications increase the population of BII substate by stabilizing the ε/ζ = g-/t state and renders higher values for the helical twist in the tested systems. In combination with the χ modification for DNA (χ*OL4*, [42]), the force field generates structures that suggest a better agreement with NMR data. The reader should pay careful attention to the use of the χ modifications, since the naming convention of the authors is the same for RNA and DNA.

The combination of the three dihedral updates (ε/ζOL1+χOL4+βOL1) are now termed OL15 [48], which are available sourcing the file *leaprc.nucleic.OL15*. More details about the OL15 force field development and test cases is available in `http://fch.upol.cz/ff_ol/`.

In a parallel effort, the group at the Barcelona Supercomputing Center have updated the well-known *bsc0* modification, now termed *bsc1*.[47] This updated version of the *bsc0* modification has also been developed using implicit solvation model and rigorous QM methodology. As with the OL15 variant, the updated *bsc1* force field increases the helical twist and yields double stranded DNA structures that are in better agreement with experimental structures. Testing of the *bsc1* force field has been performed using more than 130 systems, including single and double stranded DNA, hairpin structures, DNA-protein complexes, G-quadruplexes and more. This can be accessed by sourcing *leaprc.nuclic.bsc1*; additional information about the bsc1 force field development and test cases is available in `http://mmb.irbbarcelona.org/ParmBSC1/`.

Details of the different modifications available for DNA are presented in Table 3.2.

Regarding the performance of OL15 and bsc1 for DNA, preliminary testing comparing both force fields strongly suggest that both variations perform in a similar way and are improvements over the previous commonly bsc0 modification.[48] We refer the reader to the original articles of each force field to better understand the details and performance between each variant.

3.2.3. Some nonstandard situations

Nucleic acid residues use the new (version 3) PDB nomenclature: “DC” is used for deoxy-cytosine, and “C” for cytosine in RNA, etc. Earlier force fields (which are *not* recommended!) use “RC” for the RNA version. If you want a single, nucleoside, use “CN”, etc. For a single nucleotide, use the following command in LEaP:

Name	Modification	Notes
ff94	Original force field file	Obsolete
ff98	Modified charge set	Obsolete
ff99	Updated charge set	Foundation for all current ff's
bsc0	Barcelona α/γ backbone modification	[35]
χOL3	χmodification tuned for RNA	[37, 38]
ε/ζ OL1	ε/ζ modification for DNA	improvement for DNA, no effects for RNA [45]
χ OL4	χ modification tuned for DNA	[42]
β OL1	β dihedral modification tuned for DNA	improvement for DNA, no effects for RNA[46]
bsc1	Major update to bsc0	[47]

Table 3.2.: *Force field name and modifications for simulating nucleic DNA.*

```
cnuc = sequence { OHE C3 }
```

and analogs for other bases. Note that this will construct a protonated 5' phosphate group, which may not be what you want.

Some RNA molecules may have a 5' residue with an attached phosphate group. This requires a bit-of hand-editing of your PDB file. Suppose your 5' end looks like this (taken from PDB code 2DXI):

```
ATOM      1  OP3   G C 501      19.050  87.190  73.029  1.00 73.49           O
ATOM      2  P     G C 501      18.499  87.676  71.706  1.00 75.79           P
ATOM      3  OP1   G C 501      16.984  87.888  71.715  1.00 73.44           O
ATOM      4  OP2   G C 501      18.979  86.828  70.515  1.00 77.51           O
ATOM      5  O5'   G C 501      19.153  89.150  71.502  1.00 63.81           O
ATOM      6  C5'   G C 501      18.729  90.260  72.301  1.00 48.63           C
```

You need to edit the first atom, changing its residue name to OHE:

```
ATOM      1  OP3 OHE C 500      19.050  87.190  73.029  1.00 73.49           O
ATOM      2  P     G C 501      18.499  87.676  71.706  1.00 75.79           P
ATOM      3  OP1   G C 501      16.984  87.888  71.715  1.00 73.44           O
ATOM      4  OP2   G C 501      18.979  86.828  70.515  1.00 77.51           O
ATOM      5  O5'   G C 501      19.153  89.150  71.502  1.00 63.81           O
ATOM      6  C5'   G C 501      18.729  90.260  72.301  1.00 48.63           C
```

Note that this is not necessarily optimal: the 5' terminal phosphate will have the same charges as the phosphate in a phosphodiester linkage *between* residues along the chain. If the properties of the 5' terminal group are especially important to you, you may need to construct a special residue here. Also note (as noted above), this constructs a protonated terminal phosphate (net charge of -1); again you will need to construct special residues it you wish to have a deprotonated phosphate at the 5' position.

3.3. Carbohydrates

GLYCAM06 is a consistent and transferable parameter set for modeling carbohydrates,[49] and glycoconjugates.[50, 51] The core philosophy of the force field development process is that parameters should be: (1) be transferable to all carbohydrate ring formations and sizes, (2) be self-contained and therefore readily transferable to many quadratic force fields, (3) not require specific atom types for α- and β-anomers, (4) be readily extendible to carbohydrate derivatives and other biomolecules, (5) be applicable to monosaccharides and complex oligosaccharides, and (6) be rigorously assessed in terms of the relative accuracy of its component terms.

When combining GLYCAM06 with AMBER parameters for other biomolecules, parameter orthogonality is ensured by assigning unique atom types for GLYCAM. In order to facilitate combining GLYCAM06 with other AMBER parameter sets for other biomolecules, a variation on the GLYCAM atom types has been introduced in which the new name consists of an uppercase letter followed by second character, either a number or lowercase letter. For example the GLYCAM "CG" atom type has been changed to "Cg"; "HO" is now represented as "Ho", and so forth.

As soon as new parameters are generated, or alterations are made to existing parameters, a new version of GLYCAM is released. Updated versions that introduce new functionality are denoted using a letter suffix (i.e. GLYCAM06a, 06b, etc.). Each release is accompanied with an associated text file that summarizes the new functionality or alteration. For example, a particularly important update, released in GLYCAM06e, altered the endo-anomeric torsion term (Cg-Os-Cg-Os) in order to more accurately reproduce the populations arising from ring flips ($^4C_1$ to $^1C_4$ etc.). This particular case suggested the need to be able to independently characterize the exo- and endo-anomeric effect, which was achieved by assigning different atom types (Oa and Oe) to represent the endo-anomeric and exo-anomeric oxygen atoms, respectively.

In another important update (GLYCAM06g), a small van der Waals term was applied to all hydroxyl hydrogen atoms (Ho) to address a rare, but catastrophic, situation that can arise during MD simulations. In certain carbohydrate (and potentially other) configurations, a hydroxyl proton may be structurally constrained to being very close to a carboxylate moiety. During an MD simulation of such a system, an oscillatory motion can begin between the hydroxyl proton and the negative charge site, leading ultimately to failure of the simulation as the proton collapses onto the negatively charged moiety. The small van der Waals term (Ho, R* = 0.2000 Å, ε = 0.0300 kcal/mol) is just large enough to add sufficient repulsion to prevent this behavior, while not being large enough to perturb properties such as hydrogen bond lengths.

The GLYCAM force field family, especially, GLYCAM06, has been extensively employed in simulations of biomolecules by the larger scientific community.[52–55] The updated GLYCAM parameters and documentation are available for download at the GLYCAM-Web site (*www.glycam.org*). Also available on the website are tools for simplifying the generation of structure and topology files for performing simulations of oligosaccharides, glycoconjugates and glycoproteins. GLYCAM-Web has been integrated into several glycomics databases, such as the Consortium for Functional Glycomics (*www.functionalglycomics.org*).

GLYCAM06 force field

Always check glycam.org/params for more recent versions and new functionalities.

```
leaprc.GLYCAM_06j-1      LEaP configuration file for use of GLYCAM06
                         with carbohydrates alone or in combination
                         with the ff14SB force field.
GLYCAM_06j.dat           Parameters for oligosaccharides
GLYCAM_06j-1.prep          Structures and charges for glycosyl residues
GLYCAM_lipids_06h.prep   Structures and charges for some lipid residues
GLYCAM_amino_06j_12SB.lib    Glycoprotein libraries compatible with ff14SB.
GLYCAM_aminoct_06j_12SB.lib
GLYCAM_aminont_06j_12SB.lib
```

GLYCAM06EP force field using lone pairs (extra points)

```
GLYCAM_06EPb.dat         Parameters for oligosaccharides
GLYCAM_06EPb.prep        Structures and charges for glycosyl residues
leaprc.GLYCAM_06EPb      LEaP configuration file for GLYCAM-06EP
```

GLYCAM Force Field Parameters Download Page

```
http://www.glycam.org/params
```

GLYCAM_06j-1.prep contains prep entries for all carbohydrate residues and GLYCAM_lipids_06h.prep contains prep entries for some lipid residues (although for lipid membrane simulations we recommend you use the Amber Lipid 17 force field). GLYCAM_06EPb.prep contains prep entries for all carbohydrate residues available for modeling with extra points.

For linking glycans to proteins, libraries containing modified amino acid residues (Ser, Thr, Hyp, and Asn) must be loaded. To build a glycoprotein using ff14SB, GLYCAM_amino_06j_12SB.lib GLYCAM_aminont_06j_12SB.lib

Version	Release Date	Contributors	Change Summary
j	15 Feb., 2014	BLF	*Modified all parameters to be compatible with ff14SB. These files may not be compatible with older protein and nucleic acid force fields.*
i	27 Aug., 2013	AKN	Added two new monosaccharides to the prep file.
h	20 Oct., 2010	MBT, BLF	**Changed atom type naming to be orthogonal to other force fields.* Added HO van der Waals parameters. Set protein-related parameter values to their parm99 counterparts. Updated N-sulfation parameters.
g	20 Oct., 2010	MBT	* 1,4-scaling terms added to parameter file. Angle and torsion updates for pyranose rings, N-sulfate, phosphate and sialic acid.
f	3 Feb., 2009	MBT	* Corrected a typo in O-Acetyl term
e	28 May, 2008	MBT	* Updated glycosidic linkage terms to optimize ring puckering in pyranoses
d	12 May, 2008	SPK, MBT, ABY	Terms for thiol glycosidic linkages
c	21 Feb., 2008	MBT, ABY	* Additional (published) terms for some lipid simulations[56]
b	10 Jan., 2008	MBT, ABY	Alkanes, alkenes, amide and amino groups for some lipid simulations[56]
a	24 Apr., 2005	ABY	Sulfates & phosphates for carbohydrates

Table 3.3.: *Version change summary for the GLYCAM-06 force field. *Previously released parameters were changed. See full release notes at glycam.org/params. SPK: Sameer P. Kawatkar. MBT: Matthew B. Tessier. ABY: Austin B. Yongye. BLF: B. Lachele Foley. AKN: Anita K. Nivedha*

and GLYCAM_aminoct_06j_12SB.lib must be loaded and the desired protein force field must also be loaded. Amino acid libraries designed for linking carbohydrates modeled with extra points are not currently available.

3.3.1. File versioning

Beginning on 15 September, 2011, a new versioning system was implemented for Glycam parameters. Files produced before that date will not necessarily conform to the new system. In the new system, all files containing parameters are versioned. Users should check their contents and replace them with recent versions as appropriate.

The new versioning system employs letters and numbers. If a parameter set contains new functionality (e.g., the addition of new parameters) or fundamental changes (e.g., atom type name reassignments), a letter will be appended to its name. If the new version contains corrections (e.g., for typographical errors), its name will be appended with a number. See glycam.org/params for more documentation and examples.

Researchers are also encouraged to read the version change documentation available on the GLYCAM Parameters download page under "Documents." In this document, the changes specific to each version release are detailed. The changes are also summarized here in Table 3.3.

3.3.2. Atom type name changes

Beginning with versions g, Glycam atom type names will adopt a standard designed to keep them from overlapping with other force fields. In most cases, Glycam's type names will consist of two characters, one upper-case followed by one lower-case. Because of this, leaprc files, lib files and prep files from versions prior to g will be incompatible with current versions.

Note that some type names will not reflect the new Glycam type standard, despite being present in the Glycam force field files, for example in the files for linking glycans to amino acid residues. In these cases, Glycam will use the type name appropriate to the external force field. Parameters will be introduced only to the extent necessary to provide a link between the force fields. Since the associated parameters will also include Glycam types, they should only affect the intersections between the two force fields.

Beginning with versions j, atom type names for linking to amino acids are compatible with ff14SB. Older versions of protein and nucleic acid force fields might not be compatible.

3.3.3. General information regarding parameter development

In GLYCAM-06,[49] the torsion terms have now been entirely developed by fitting to quantum mechanical data (B3LYP/6-31++G(2d,2p)//HF/6-31G(d)) for small-molecules. This has converted GLYCAM-06 into an additive force field that is extensible to diverse molecular classes including, for example, lipids and glycolipids. The parameters are self-contained, such that it is not necessary to load any AMBER parameter files when modeling carbohydrates or lipids. To maintain orthogonality with AMBER parameters for proteins, notably those involving the CT atom type, tetrahedral carbon atoms in GLYCAM are called Cg (C-GLYCAM, CG in previous releases). Thus, GLYCAM and AMBER may be combined for modeling carbohydrate-protein complexes and glycoproteins. More information on atom type names is available in 3.3.2 . Because the GLYCAM-06 torsion terms were derived by fitting to data for small, often highly symmetric molecules, asymmetric phase shifts were not required in the parameters. This has the significant advantage that it allows one set of torsion terms to be used for both α- and β-carbohydrate anomers regardless of monosaccharide ring size or conformation. A molecular development suite of more than 75 molecules was employed, with a test suite that included carbohydrates and numerous smaller molecular fragments. The GLYCAM-06 force field has been validated against quantum mechanical and experimental properties, including: gas-phase conformational energies, hydrogen bond energies, and vibrational frequencies; solution-phase rotamer populations (from NMR data); and solid-phase vibrational frequencies and crystallographic unit cell dimensions.

3.3.4. Scaling of electrostatic and nonbonded interactions

As in previous versions of GLYCAM,[2] the parameters were derived for use without scaling 1-4 non-bonded and electrostatic interactions. Thus, in *sander*, *pmemd*, and so on, the simulation parameters *scnb* and *scee* should typically be set to unity. We have shown that this is essential in order to properly treat internal hydrogen bonds, particularly those associated with the hydroxymethyl group, and to correctly reproduce the rotamer populations for the C5-C6 bond.[57] Beginning with Amber 11, it is now possible to employ mixed scaling of the *scnb* and *scee* parameters. Anyone wishing to simulate systems containing both carbohydrates and proteins should use the new mixed scaling capability. To do this, any scaling factors that differ from the default must be included in the parameter file. Beginning with the GLYCAM_06g parameter file shipped with Amber 11, these factors are already included. Anyone wishing to employ earlier parameter sets must modify the files.

3.3.5. Development of partial atomic charges

As in previous versions of GLYCAM, the atomic partial charges were determined using the RESP formalism, with a weighting factor of 0.01,[49, 58] from a wavefunction computed at the HF/6-31G(d) level. To reduce artifactual fluctuations in the charges on aliphatic hydrogen atoms, and on the adjacent saturated carbon atoms, charges on aliphatic hydrogens (types HC, H1, H2, and H3) were set to zero while the partial charges were fit to the remaining atoms.[59] It should be noted that aliphatic hydrogen atoms typically carry partial charges that fluctuate around zero when they are included in the RESP fitting, particularly when averaged over conformational ensembles.[49, 60] In order to account for the effects of charge variation associated with exocyclic bond rotation, particularly associated with hydroxyl and hydroxylmethyl groups, partial atomic charges for each sugar were determined by averaging RESP charges obtained from 100 conformations selected evenly from 10-50 ns solvated MD simulations of the methyl glycoside of each monosaccharide, thus yielding an ensemble averaged charge set.[49, 60]

Carbohydrate	Pyranose	Furanose
	α/β, D/L	α/β, D/L
Arabinose	yes	yes
Lyxose	yes	yes
Ribose	yes	yes
Xylose	yes	yes
Allose	yes	
Altrose	yes	
Galactose	yes	*a*
Glucose	yes	*a*
Gulose	yes	
Idose	*a*	
Mannose	yes	
Talose	yes	
Fructose	yes	yes
Psicose	yes	yes
Sorbose	yes	yes
Tagatose	yes	yes
Fucose	yes	
Quinovose	yes	
Rhamnose	yes	
Galacturonic Acid	yes	
Glucuronic Acid	yes	
Iduronic Acid	yes	
N-Acetylgalactosamine	yes	
N-Acetylglucosamine	yes	
N-Acetylmannosamine	yes	
Neu5Ac	yes, *b*	yes,*b*
KDN	*a*,*b*	*a*,*b*
KDO	*a*,*b*	*a*,*b*

Table 3.4.: *Current Status of Monosaccharide Availability in GLYCAM. (a) Currently under development. (b) Only one enantiomer and ring form known.*

3.3.6. Carbohydrate parameters for use with the TIP5P water model

In order to extend GLYCAM to simulations employing the TIP-5P water model, an additional set of carbohydrate parameters, GLYCAM-06EP, has been derived in which lone pairs (or extra points, EPs) have been incorporated on the oxygen atoms.[61] The optimal O-EP distance was located by obtaining the best fit to the HF/6-31g(d) electrostatic potential. In general, the best fit to the quantum potential coincided with a negligible charge on the oxygen nuclear position. The optimal O-EP distance for an sp3 oxygen atom was found to be 0.70 Å; for an sp2 oxygen atom a shorter length of 0.3 Åwas optimal. When applied to water, this approach to locating the lone pair positions and assigning the partial charges yielded a model that was essentially indistinguishable from TIP-5P. Therefore, we believe this model is well suited for use with TIP-5P.[61] The new files are named 06EP (originally 04EP), as they have been corrected for numerous typographical errors and updated to match current naming and residue structure conventions.

3.3.7. Carbohydrate Naming Convention in GLYCAM

In order to incorporate carbohydrates in a standardized way into modeling programs, as well as to provide a standard for X-ray and NMR protein database files (pdb), we have developed a three-letter code nomenclature. The restriction to three letters is based on standards imposed on protein data bank (PDB) files by the RCSB PDB Advisory Committee (*www.rcsb.org/pdb/pdbac.html*), and for the practical reason that all modeling and experimental software has been developed to read three-letter codes, primarily for use with protein and nucleic acids.

As a basis for a three-letter PDB code for monosaccharides, we have introduced a one-letter code for monosaccharides (Table 3.5).[62] Where possible, the letter is taken from the first letter of the monosaccharide name. Given the endless variety in monosaccharide derivatives, the limitation of 26 letters ensures that no one-letter (or three-letter) code can be all encompassing. We have therefore allocated single letters firstly to all 5- and 6-carbon, non-derivatized monosaccharides. Subsequently, letters have been assigned on the order of frequency of occurrence or biological significance.

Using three letters (Tables 3.6 to 3.8), the present GLYCAM residue names encode the following content: carbohydrate residue name (Glc, Gal, etc.), ring form (pyranosyl or furanosyl), anomeric configuration (α or β, enantiomeric form (D or L) and occupied linkage positions (2-, 2,3-, 2,4,6-, etc.). Incorporation of linkage position is a particularly useful addition, since, unlike amino acids, the linkage cannot otherwise be inferred from the monosaccharide name. Further, the three-letter codes were chosen to be orthogonal to those currently employed for amino acids.

	Carbohydrate[a]	One letter code[b]	Common Abbreviation
1	D-Arabinose	A	Ara
2	D-Lyxose	D	Lyx
3	D-Ribose	R	Rib
4	D-Xylose	X	Xyl
5	D-Allose	N	All
6	D-Altrose	E	Alt
7	D-Galactose	L	Gal
8	D-Glucose	G	Glc
9	D-Gulose	K	Gul
10	D-Idose	I	Ido
11	D-Mannose	M	Man
12	D-Talose	T	Tal
13	D-Fructose	C	Fru
14	D-Psicose	P	Psi
15	D-Sorbose	B[d]	Sor
16	D-Tagatose	J	Tag
17	D-Fucose (6-deoxy D-galactose)	F	Fuc
18	D-Quinovose (6-deoxy D-glucose)	Q	Qui
19	D-Rhamnose (6-deoxy D-mannose)	H	Rha
20	D-Galacturonic Acid	O[d]	GalA
21	D-Glucuronic Acid	Z[d]	GlcA
22	D-Iduronic Acid	U[d]	IdoA
23	D-*N*-Acetylgalactosamine	V[d]	GalNac
24	D-*N*-Acetylglucosamine	Y[d]	GlcNAc
25	D-*N*-Acetylmannosamine	W[d]	ManNAc
26	*N*-Acetyl-neuraminic Acid	S[d]	NeuNAc, Neu5Ac
	KDN	KN[c,d]	KDN
	KDO	KO[c,d]	KDO
	N-Glycolyl-neuraminic Acid	SG[c,d]	NeuNGc, Neu5Gc

Table 3.5.: *The one-letter codes that form the core of the GLYCAM residue names for monosaccharides* [a]*Users requiring prep files for residues not currently available may contact the Woods group (*www.glycam.org*) to request generation of structures and ensemble averaged charges.* [b]*Lowercase letters indicate* L-*sugars, thus* L-*Fucose would be "f", see Table 3.8 .* [c]*Less common residues that cannot be assigned a single letter code are accommodated at the expense of some information content.* [d]*Nomenclature involving these residues will likely change in future releases.[62] Please visit* www.glycam.org *for the most updated information.*

	$\alpha-$D-Glc*p*	$\beta-$D-Gal*p*	$\alpha-$D-Ara*p*	$\beta-$D-Xyl*p*
Linkage Position	Residue Name	Residue Name	Residue Name	Residue Name
Terminal[b]	0GA[b]	0LB	0AA	0XB
1-[c]	1GA[c]	1LB	1AA	1XB
2-	2GA	2LB	2AA	2XB
3-	3GA	3LB	3AA	3XB
4-	4GA	4LB	4AA	4XB
6-	6GA	6LB		
2,3-	ZGA[d]	ZLB	ZAA	ZXB
2,4-	YGA	YLB	YAA	YXB
2,6-	XGA	XLB		
3,4-	WGA	WLB	WAA	WXB
3,6-	VGA	VLB		
4,6-	UGA	ULB		
2,3,4-	TGA	TLB	TAA	TXB
2,3,6-	SGA	SLB		
2,4,6-	RGA	RLB		
3,4,6-	QGA	QLB		
2,3,4,6-	PGA	PLB		

Table 3.6.: *Specification of linkage position and anomeric configuration in* D*-hexo- and* D*-pentopyranoses in three-letter codes based on the GLYCAM one-letter code* [a]*In pyranoses A signifies* α*-configuration; B =* β*.* [b]*Previously called GA, the zero prefix indicates that there are no oxygen atoms available for bond formation, i.e., that the residue is for chain termination.* [c]*Introduced to facilitate the formation of a 1–1´ linkage as in* α*-*D*-Glc-1-1´-*α*-*D*-Glc {1GA 0GA}.* [d]*For linkages involving more than one position, it is necessary to avoid employing prefix letters that would lead to a three-letter code that was already employed for amino acids, such as ALA.*

	α-D-Glc*f*	β-D-Man*f*	α-D-Ara*f*	β-D-Xyl*f*
Linkage position	Residue name	Residue name	Residue name	Residue name
Terminal	0GD	0MU	0AD	0XU
1-	1GD	1MU	1AD	1XU
2-	2GD	2MU	2AD	2XU
3-	3GD	3MU	3AD	3XU
…	…	…	…	…
etc.	etc.	etc.	etc.	etc.

Table 3.7.: *Specification of linkage position and anomeric configuration in* D*-hexo- and* D*pentofuranoses in three-letter codes based on the GLYCAM one-letter code. In furanoses D (down) signifies* α*; U (up) =* β*.*

	α-L-Glc*p*	β-L-Man*p*	α-L-Ara*p*	β-L-Xyl*p*
Linkage position	Residue name	Residue name	Residue name	Residue name
Terminal	0gA	0mB	0aA	0xB
1-	1gA	1mB	1aA	1xB
2-	2gA	2mB	2aA	2xB
3-	3gA	3mB	3aA	3xB
…	…	…	…	…
etc.	etc.	etc.	etc.	etc.

Table 3.8.: *Specification of linkage position and anomeric configuration in* L*-hexo- and* L*pentofuranoses in three-letter codes.*

3.4. Lipids

Biological processes in the human body are dependent on highly specific molecular interactions. The vast majority of the interactions take place in compartments within the cell, and an understanding of the behavior of the membranes that compartmentalize and enclose the cell is therefore critical for rationalizing these processes. Biological membranes are complex structures formed mostly by lipids and proteins. For this reason lipid bilayers have received a lot of attention both computationally and experimentally for many years.[63, 64] The vital role of cell membranes is underlined by the estimation that over half of all proteins interact with membranes, either transiently or permanently.[65] Further, G protein-coupled receptors embedded in the membrane account for 50–60% of present day drug targets, and membrane proteins as a whole make up around 70%.[66] Even so, only 685 resolved unique structures of membrane embedded proteins, out of a total of 65 500 searchable entries (after removing redundant structures), exist in the Protein Data Bank (April 2017) reflecting the difficulties in studying membrane-associated proteins experimentally, making them prime targets for simulation.

Prior to 2012, the only force field parameters for lipids distributed with AmberTools were part of the Glycam force field and were limited in scope.[56] Traditionally, lipid simulations with Amber have either employed the Charmm parameters, via support for the Charmm force fields through the Chamber package[67] or through attempts to adapt the General Amber Force Field (GAFF) with limited success.

In 2012, Amber greatly expanded support for simulation of lipids. This includes the development of a modular framework for lipid simulations and initial parameterization within the *LIPID11* force field[68] as well as a careful refinement of the non-bonded parameters and associated torsion terms within the GAFF force field for specific application to lipids.[69] The latter, *GAFFLipid,* was the first lipid parameter set based on the Amber force field equation to support simulation of lipid bilayers in the tensionless NPT ensemble while the former, LIPID11, provided the first modular framework for constructing lipid simulations analogous to the Amber amino and nucleic acid force fields. Together these developments have made simulation of phospholipids with AMBER substantially easier. LIPID14 was released in 2014 [70] and represented a major advancement over the previous Amber compatible lipid force fields for lipid bilayer simulations in the NPT ensemble without the need for an artificial constant surface tension term. Validation of the LIPID14 parameters were provided through extensive self-assembly simulations [71, 72]. Inclusion and validation of parameters for cholesterol [73] represented an important addition to the lipid parameter set, allowing even more complex lipid containing systems to be simulated. LIPID17 is the most recent release of Amber lipid parameters. It builds upon the modularity of LIPID14 and provides an extension of modular phospholipid residues to include anionic head groups and polyunsaturated tails. In the process the bonded alkane parameters have been revised and updated by fitting to quantum energies. Furthermore, new partial charges have been generated for all the head group residues in order to accommodate the anionic head groups whilst maintaining consistency in the charge derivation approach. Details regarding the parameterization are given in Skjevik et al. (####). The modular nature of the force field allows for many combinations of lipid head and tail groups as well as rapid and standardized parameterization of additional lipids. LIPID17 was validated through bilayer simulations of eighteen different phospholipid types, for a total of 0.6 microseconds each without applying a surface tension or constant area term. The lipid bilayer structural features compare favorably with experimental measures such as area per lipid, bilayer thickness, NMR order parameters, scattering data, and lipid lateral diffusion.

3.4.1. LIPID17: The Amber lipid force field

```
leaprc.lipid17     defines atom types and loads the files below
lipid17.lib        atoms, charges, and topologies for LIPID17 residues
lipid17.dat        LIPID17 force field parameters
```

The LIPID17 force field represented the logical next step in the development of an Amber lipid force field that build on the modular nature of LIPID11[68] and LIPID 14 [70] to allow for tensionless lipid bilayer simulations in Amber. LIPID17 (####) has been designed to be fully compatible with the other pairwise-additive protein, nucleic acid, carbohydrate, and small molecule Amber force fields.

LIPID17 is a modular force field for the simulation of phospholipids and cholesterol. To achieve this modularity phospholipids are divided into interchangeable head group and tail group "residues."

	Description	LIPID17 Residue Name
Acyl chain	Lauroyl (12:0)	LAL
	Myristoyl (14:0)	MY
	Palmitoyl (16:0)	PA
	Oleoyl (18:1 n-9)	OL
	Stearoyl (18:0)	ST
	Arachidonoyl (20:4)	AR
	Docosahexaenoyl (22:6)	DHA
Head group	Phosphatidylcholine	PC
	Phosphatidylethanolamine	PE
	Phosphatidylserine	PS
	Phosphatidylglycerol	PGR
	Phosphaditic acid	PH-
Other	Cholesterol	CHL

Table 3.9.: *LIPID17 residue names.*

Lipid 1	sn-1 tail residue
	head group residue
	sn-2 tail residue
	TER card
Lipid 2	sn-1 tail residue
	head group residue
	sn-2 tail residue
	TER card
...	...

Table 3.10.: *LIPID17 PDB format for LEaP*

Currently, there are seven tail group residues and five head group residues supported, as well as cholesterol, and LEaP supports any combination of these lipid residues. The supported LIPID17 residues and their residue names are listed in Table 3.9. LIPID17 can be used alone or in conjunction with other Amber force fields. The order with which the various AMBER force fields are loaded along with LIPID17 should not matter. For example, to load ff14SB and LIPID17 in LEaP use:

```
source leaprc.protein.ff14SB
source leaprc.lipid17
```

LIPID17 PDB format

LIPID17 atom names and types are defined in Skjevik, et al[68], Dickson, et al[70], Madej et al[73] and Skjevik et al (*####*).

A properly formatted lipid PDB can be loaded into LEaP. Each phospholipid molecule in LIPID17 is made up of three residues. Atoms from each residue must be in contiguous blocks and ordered as described below in each molecule. A TER card must be appended after all the atoms for each molecule. Table 3.10 specifies the residue format for the PDB file loaded by LEaP in order to correctly define linker atoms.

The connectivity (CONECT records) section of the PDB is redundant and should be removed prior to loading into LEaP. The head group and tail residues are linked together by the LEaP program after loading the lipid PDB file.

PDB formatted structure files with alternative residue and atom names (such as Charmm C36) may be converted to the LIPID17 naming convention by way of the script called *charmmlipid2amber.py* which is supplied with AmberTools to convert Charmm C36 residue and atom names to LIPID17 nomenclature.

```
charmmlipid2amber.py -i charmm_c36.pdb -o output lipid17.pdb
```

3.5. Solvents

```
leaprc.water.<type> loads solvents.lib and the appropriate frcmod file
solvents.lib        library for water, methanol, chloroform, NMA, urea
frcmod.tip4p        Parameter changes for TIP4P.
frcmod.tip4pew      Parameter changes for TIP4PEW.
frcmod.tip5p        Parameter changes for TIP5P.
frcmod.spce         Parameter changes for SPC/E.
frcmod.spceb        Parameter changes for SPC/Eb.
frcmod.opc          Parameter changes for OPC.
frcmod.opc3         Parameter changes for OPC3.
frcmod.pol3         Parameter changes for POL3.
frcmod.tip3pfb      Parameter changes for the force-balance TIP3P model
frcmod.tip4pfb      Parameter changes for the force-balance TIP4P model

frcmod.meoh         Parameters for methanol.
frcmod.chcl3        Parameters for chloroform.
frcmod.nma          Parameters for N-methyacetamide.
frcmod.urea         Parameters for urea (or urea-water mixtures).
```

Amber provides direct support for several water models. There is no default, but the most common water model is TIP3P.[74] This model will be used for residues with names HOH or WAT. If you want to use other water models, execute the following leap commands after loading your leaprc file:

```
WAT = PL3 (residues named WAT in pdb file will be POL3)
source leaprc.water.pol3
```

(The above is obviously for the POL3 model.) The *solvents.lib* file contains TIP3P,[74] TIP3P/F,[75] TIP4P,[74, 76] TIP4P/Ew,[77, 78] TIP5P,[79] OPC,[80] OPC3,[81], POL3[82], SPC/E[83], SPC/Eb[24], TIP3PFB[26] and TIP4PFB[26] models for water; these are called TP3, TPF, TP4, T4E, TP5, OPC, PL3, SPC, SPC, FB3 and FB4, respectively. (The SPC/E and SPC/Eb models are both called SPC: you just have to be sure to load the appropriate frcmod file.) By default, the residue name in the prmtop file will be WAT, regardless of which water model is used.

The "standard" leaprc files for *tip3p*, *spce*, *tip4pew* and *opc* also load the Joung/Cheatham monovalent ion parameters (see below). If you wish to use other parameters, or to deal with divalent or other ions, you will need to load the appropriate frcmod files.

Amber has two flexible water models, one for classical dynamics, SPC/Fw[84] (called "SPF") and one for path-integral MD, qSPC/Fw[85] (called "SPG"). You would use these in the following manner:

```
WAT = SPG
loadAmberParams frcmod.qspcfw
set default FlexibleWater on
```

Then, when you load a PDB file with residues called WAT, they will get the parameters for qSPC/Fw. (Obviously, you need to run some version of quantum dynamics if you are using qSPC/Fw water.)

The *solvents.lib* file, which is automatically loaded with many leaprc files, also contains pre-equilibrated boxes for many of these water models. These are called POL3BOX, QSPCFWBOX, SPCBOX, SPCFWBOX, TIP3PBOX, TIP3PFBOX, TIP4PBOX, TIP4PEWBOX, OPCBOX, OPC3BOX, and TIP5PBOX. These can be used as arguments to the *solvateBox* or *solvateOct* commands in LEaP.

In addition, non-polarizable models for the organic solvents methanol, chloroform and N-methylacetamide are provided,[86] along with a box for an 8M urea-water mixture. The input files for a single molecule are in *$AMBERHOME/dat/leap/prep*, and the corresponding frcmod files are in *$AMBERHOME/dat/leap/parm*. Pre-equilibrated boxes are in *$AMBERHOME/dat/leap/lib*. For example, to solvate a simple peptide in methanol, you could do the following:

```
source leaprc.protein.ff14SB (get a standard force field)
loadAmberParams frcmod.meoh (get methanol parameters)
```

```
peptide = sequence { ACE VAL NME } (construct a simple peptide)
solvateBox peptide MEOHBOX 12.0 0.8 (solvate the peptide with meoh)
saveAmberParm peptide prmtop prmcrd
quit
```

Similar commands will work for other solvent models.

3.5.1. OPC family water models

OPC is a new non-polarizable, 4-point, 3-charge rigid water model.[80] Geometrically, it resembles TIP4P-like models, although the values of OPC point charges and charge-charge distances are quite different. The model has a single VDW center on the oxygen nucleus. The model is constructed based on the concept of optimal point charge approximation; [87] the central idea of OPC is to distribute the point charges to best reproduce the 3 lowest order multipole moments of water molecule in liquid phase. The optimal values for the dipole μ and the square quadrupole moment Q_T [88] are determined as best fit values that reproduce key experimental properties of water in liquid phase. The low dimensionality of the parameter space μ-Q_T permits a virtually exhaustive search. The linear quadrupole and the octupole moments[89] are fixed to values obtained from high quality QM calculations.[88]

A full description of OPC and its properties can be found in Ref.[80]. For 11 key liquid state properties against which water models are most often benchmarked, OPC is on average within 0.76% of the experiment (relative error). This accuracy is dramatically better compared to the commonly used rigid models. For example, the dielectric constant of TIP3P and TIP4PEw is 94 and 63.9 respectively, while OPC predicts it to be 78.4±0.6 (the experimental value is 78.4). The reported OPC properties were computed using Amber 12 on GPUs with a time-step of 2 fs, periodic boundary conditions, an 8 angstrom cut-off for nonbonded interactions, and PME for long range electrostatics. SHAKE was used to constrain hydrogens. The rest of parameters are set to current Amber defaults; note that these include accounting for the van der Waals interactions beyond the cut-off via a continuum model (vdwmeth=1).

Because of the improved accuracy in bulk properties, OPC delivers noticeable accuracy improvement in practical atomistic simulations, even with existing force-fields. Improvements were reported in: RNA simulations,[90] thermodynamics of ligand binding,[91] small molecule hydration,[80] and simulations of intrinsically disordered proteins (unpublished).

Two sets of 12-6 LJ parameters for OPC water model (the 12-6 IOD set and the 12-6 HFE set) for 3 monovalent ions (Na+, K+, Cl-) have been developed by Li, Merz and co-workers, see Section 3.6 for the definition and important usage suggestions. Our tests show that the deviation of the Ion-Oxygen Distances predicted using the 12-6 HFE set from the reference IOD values is within ±0.2A. Comparing these deviations to those reported for other ion parameter sets available, it seems that the magnitude of the deviation is borderline acceptable, which means that the HFE set might also work in situations where IOD is formally recommended. For Na+ the transferability is not an issue as HFE and IOD parameters are essentially the same. In situations where agreement of HFEs with one of the common experimental references is critical, the use of OPC-specific parameters (the 12-6 HFE set) for K+, Na+, and Cl- may be advisable. The IOD parameter set are recommended to be used in the structural refinement.

Based on our limited experience, it appears that the Joung/Cheatham ion parameters for TIP4P-EW (jc_tip4pew)[92] may also be acceptable for OPC water model, especially when accurate reproduction of IODs is critical. One advantage of using the jc_tip4pew set is that it provides a consistent set of parameters for most biologically relevant ions, not just K+, Na+, and Cl-. Another advantage at the moment is that the set has already been tested in practice with OPC model.[90, 91]

OPC3 – a 3-point rigid non-polarizable water model – is the latest addition to the family, constructed using the same philosophy as OPC. Further details are available in Ref.[81]. Briefly, OPC3 is significantly more accurate than the commonly used water models of same class (TIP3P, SPC/E) in reproducing a comprehensive set of liquid bulk properties, over a wide range of temperatures. Compareed to the 4-point OPC, OPC3 is somewhat less accurate comapred to experiment. Until model-specific ion parametrs have been developed, we cautiosly recommend the Joung/Cheatham ion parameters previously developed for TIP3P to be used with OPC3.

3.6. Ions

```
frcmod.ionsjc_tip3p      Joung/Cheatham ion parameters for TIP3P water
frcmod.ionsjc_spce       same, but for SPC/E water
frcmod.ionsjc_tip4pew    same, but for TIP4P/EW water

frcmod.ions1lm_126_tip3p    Li/Merz ion parameters for +1 and -1 ions in TIP3P water (12-6 normal usage set)
frcmod.ions1lm_126_spce     same, but in SPC/E water
frcmod.ions1lm_126_tip4pew  same, but in TIP4P/EW water
frcmod.ions1lm_iod          Li/Merz ion parameters for +1 and -1 ions (12-6 IOD set)

frcmod.ions234lm_126_tip3p    Li/Merz ion parameters for +2 to +4 ions in TIP3P water (12-6 normal usage set)
frcmod.ions234lm_126_spce     same, but in SPC/E water
frcmod.ions234lm_126_tip4pew  same, but in TIP4P/EW water
frcmod.ions234lm_hfe_tip3p    Li/Merz ion parameters for +2 to +4 ions in TIP3P water (12-6 HFE set)
frcmod.ions234lm_hfe_spce     same, but in SPC/E water
frcmod.ions234lm_hfe_tip4pew  same, but in TIP4PEW water
frcmod.ions234lm_iod_tip3p    Li/Merz ion parameters for +2 to +4 ions in TIP3P water (12-6 IOD set)
frcmod.ions234lm_iod_spce     same, but in SPC/E water
frcmod.ions234lm_iod_tip4pew  same, but in TIP4P/EW water

frcmod.ions1lm_1264_tip3p   Li/Merz ion parameters for -1 and +1 ions in TIP3P water (12-6-4 set)
frcmod.ions1lm_1264_spce    same, but in SPC/E water
frcmod.ions1lm_1264_tip4pew same, but in TIP4PEW water

frcmod.ions234lm_1264_tip3p   Li/Merz ion parameters for +2 to +4 ions in TIP3P water (12-6-4 set)
frcmod.ions234lm_1264_spce    same, but in SPC/E water
frcmod.ions234lm_1264_tip4pew same, but in TIP4PEW water

frcmod.ions1lm_126_hfe_opc   Li/Merz ion parameters for +1 and -1 ions in OPC water (12-6 HFE set)
frcmod.ions1lm_126_iod_opc   Li/Merz ion parameters for +1 and -1 ions in OPC water (12-6 IOD set)

atomic_ions.lib          topologies for monoatomic ions (new naming scheme)
ions94.lib               topologies for ions with the old naming scheme
```

In 2008, Joung and Cheatham created a consistent set of parameters for alkali halide ions, fitting solvation free energies, radial distribution functions, ion-water interaction energies and crystal lattice energies and lattice constants for non-polarizable spherical ions.[92, 93] These have been separately parametrized for each of three popular water models, as indicated above.

Li, Merz and co-workers subsequently developed ion parameters for the monovalent, divalent, trivalent and tetravalent ions for the 12-6 LJ nonbonded model and the 12-6-4 LJ-type nonbonded model for PME simulations.[94–97] The experimental values they tried to reproduce are the experimental Hydration Free Energy (HFE) values, Ion-Oxygen Distance (IOD) values and Coordination Number (CN) values of the first solvation shell. It was found that it is hard to reproduce the three experimental values simultaneously by using the 12-6 LJ nonbonded model. Since the charge-induced dipole interaction is proportional tor^{-4} , a new term with format $(C/r)^4$ was added to the 12-6 LJ potential, yielding a 12-6-4 LJ-type potential. The new potential with designed parameters could reproduce the experimental HFE, IOD and CN values at the same time without significant compromise. Especially for the highly charged metal ions, the 12-6-4 LJ-type nonbonded model performs much better than the 12-6 one overall. Similar to Joung and Cheatam's work, three water models were treated separately for the parameter design, as indicated in the name of frcmod files. Users can check the notes in the frcmod files to see the reference of each parameter.

For the 12-6 LJ nonbonded model, three different parameter sets are available for each water model to meet different requirements:

1. 12-6 normal usage set. In which we have HFE set of the monovalent ions (could reproduce the experimental HFE),[97] the Compromise (CM) set of divalent ions (which could reproduce the experimental relative HFE and CN values),[95] and IOD set (could reproduce the experimental IOD) for the trivalent and tetravalent ions.[96] **These parameters are recommended to be used in the normal MD simulations.** This is because

for the monovalent ions the error of the 12-6 LJ nonbonded model is pretty small (a CM set may not needed since HFE or IOD set are pretty close to each other) while for the trivalent and tetravalent metal ions the 12-6 LJ nonboned model has relative big error (a CM set could have big error for both HFE and IOD at this moment).

2. 12-6 HFE set to reproduce experimental HFE.[94, 96, 97] The HFE parameter set has limited error for monovalent ions, while could have remarkable error for highly charged ions. Since we use the HFE set for monovalent ions in the 12-6 normal usage set, so we don't have a sepecific HFE set parameter file for monovalent ions.

3. 12-6 IOD set to reproduce experimental IOD.[94, 96, 97] Since the ion with certain parameter could reproduce similar IOD values in the three water models, so the IOD set parameters of three water models were designed identical (for the monovalent and divalent metal ions, while for the trivalent and tetravalent ions, the IOD set are estimated for each water model separately). **The IOD parameter set are recommended to be used in the structural refinement or for structural property orientated investigation.**

For the 12-6-4 LJ-type nobonded model, only one parameter set (12-6-4 set) designed for each of the three water models. The 12-6-4 model has also been tested in mixed systems (such as nucleic acids, proteins and ionic solutions) and have shown excellent transferability.[95–97] In the recent work of Panteva *et al.* found the 12-6-4 model gives improved results over the 12-6 model when modeling Mg^{2+} with nucleic acid systems. Additionally, the 12-6-4 model with the SPC/E water model performed exceptionally well for simulating all properties in these benchmark calculations.[98] The VDW parameters which specified designed for the divalent metal ions with 12-6-4 LJ-type nonbonded model are shown as the 12-6-4 set in the above. These frcmod files can be used to generate an original prmtop file. After obtaining the original prmtop file, you can use the *add12_6_4* command in *parmed* to generate a prmtop with the additionalC_4 terms with the flag `LENNARD_JONES_CCOEF`. Please see the add12_6_4 command14.2.2.6 in Subsection14.2.2 in the manual for detailed information. After obtaining the prmtop with the additional C_4 term, you can use *sander* or *pmemd* to run the simulation. Recently Penteva et al. fine-tuned theC_4 terms between several divalent metal ions (Mg^{2+}, Mn^{2+}, Zn^{2+}, and Cd^{2+}) and nucleic acid systems[99] while keep theC_4 terms between metal ions and water desgined by Li and Merz.[95] The new parameter set could better balance the interaction types in the nucleic acid systems and are recommended to use in related modeling. An related tutorial is shown in the following webpage: "http://ambermd.org/tutorials/advanced/tutorial20/12_6_4.htm".

3.7. Modified amino acids and nucleotides

Parameters for phosphorylated amino acids [100, 101] can be obtained by typing "`source leaprc.phosaa10`" in your leap.in file. Many post-translational modifications are also available at *http://selene.princeton.edu/FFPTM/*. Parameters for common modifications for RNA nucleotides [44]can by loaded with "`source leaprc.modrna08`". Pointers to other sets of Amber-compatible force fields may be found at the Amber web site, *http://ambermd.org/*.

Additional parameters for six common fluorescent protein chromophores–eGFP, eBFP, eYFP, eCFP, DsRed, and mCherry–are available[102] by sourcing *leaprc.xFPchromophores* after sourcing the main force field leaprc file (e.g. *leaprc.protein.ff14SB*). This will allow seamless loading of PDB files containing fluorescent proteins provided they follow standard naming of the chromophore: eGFP=CRO, eBFP=IIC, eYFP=CR2, eCFP=CRF, DsRed= CRQ, and mCherry=CH6. The chromophore parmaeters are based on parm10 with the ff14SB modifications, but also borrow heavily from GAFF. Both uppercase and lowercase atom types are utilized, so users should take caution if mixing ff14SB with GAFF. See original reference[102] for details of implementation.

3.8. Force fields related to semi-empirical QM

ParmAM1 and **parmPM3** are classical force field parameter sets that reproduce the geometry of proteins minimized at the semi-empirical AM1 or PM3 level, respectively.[103] These new force fields provide an inexpensive, yet reliable, method to arrive at geometries that are more consistent with a semi-empirical treatment of protein structure. These force fields are meant only to reproduce AM1 and PM3 geometries (warts and all) and were not tested for use in other instances (e.g., in classical MD simulations, etc.) Since the minimization of a protein structure at the semi-empirical level can become cost-prohibitive, a “preminimization” with an appropriately parametrized classical treatment will facilitate future analysis using AM1 or PM3 Hamiltonians.

3.9. Obsolete force field files

The following files are included for historical interest. We do *not* recommend that these be used any more for molecular simulations. The leaprc files that load these files have been moved to *$AMBERHOME/dat/leap/parm/oldff*.

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

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

The **ff86** parameters are described in early papers from the Kollman and Case groups.[104, 105] [The “parm91” designation is somewhat unfortunate: this file is really only a corrected version of the parameters described in the 1984 and 1986 papers listed above.] These parameters are not generally recommended any more, but may still be useful for vacuum simulations of nucleic acids and proteins using a distance-dependent dielectric, or for comparisons to earlier work. The material in *parm91X.dat* is the parameter set distributed with Amber 4.0. The *STUB* nonbonded set has been copied from *parmuni.dat*; these sets of parameters are appropriate for united atom calculations using the “larger” carbon radii referred to in the “note added in proof” of the 1984 JACS paper. If these values are used for a united atom calculation, the parameter *scnb* must be defined in the prmtop file and should be set to 8.0; for all-atom calculations it should be 2.0. The *scee* parameter should be defined in the prmtop file and set to 2.0 for both united atom and all-atom variants. *Note that the default value for scee is now 1.2 (the value for 1994 and later force fields); this must be explicitly defined in the prmtop file when using the earlier force fields.*

parm91X.dat is not recommended. However, for historical completeness a number of terms in the non-bonded list of *parm91X.dat* should be noted. The non-bonded terms for I (iodine), CU (copper) and MG (magnesium) have not been carefully calibrated, but are given as approximate values. In the *STUB* set of non-bonded parameters, we have included parameters for a large hydrated monovalent cation (IP) that represent work by Singh *et al.*[106] on large hydrated counterions for DNA. Similar values are included for a hydrated anion (IM).

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

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

`all_nuc94.in`	`Nucleic acid input for building database.`
`all_amino94.in`	`Amino acid input for building database.`
`all_aminoct94.in`	`COO- amino acid input for database.`
`all_aminont94.in`	`NH3+ amino acid input for database.`
`nacl.in`	`Ion file.`
`parm94.dat`	`1994 force field file.`
`parm96.dat`	`Modified version of 1994 force field, for proteins.`
`parm98.dat`	`Modified version of 1994 force field, for nucleic acids.`

Contained in **ff94** are parameters from the so-called "second generation" force field developed in the Kollman group in the early 1990s.[25] These parameters are especially derived for solvated systems, and when used with an appropriate 1-4 electrostatic scale factor, have been shown to perform well at modeling many organic molecules. The parameters in *parm94.dat* omit the hydrogen bonding terms of earlier force fields. This is an all-atom force field; no united-atom counterpart is provided. 1-4 electrostatic interactions are scaled by 1.2 instead of the value of 2.0 that had been used in earlier force fields.

Charges were derived using Hartree-Fock theory with the 6-31G* basis set, because this exaggerates the dipole moment of most residues by 10-20%. It thus "builds in" the amount of polarization which would be expected in aqueous solution. This is necessary for carrying out condensed phase simulations with an effective two-body force field which does not include explicit polarization. The charge-fitting procedure is described in Ref [25].

The **ff96** force field [107] differs from *parm94.dat* in that the torsions for ϕ and ψ have been modified in response to *ab initio* calculations [108] which showed that the energy difference between conformations were quite different than calculated by Cornell *et al.* (using *parm94.dat*). To create *parm96.dat*, common V1 and V2 parameters were used for ϕ and ψ, which were empirically adjusted to reproduce the energy difference between extended and constrained alpha helical energies for the alanine tetrapeptide. This led to a significant improvement between molecular mechanical and quantum mechanical relative energies for the remaining members of the set of tetrapeptides studied by Beachy *et al.* Users should be aware that *parm96.dat* has not been as extensively used as *parm94.dat*, and that it almost certainly has its own biases and idiosyncrasies, including strong bias favoring extended β conformations.[17, 109, 110]

The **ff98** force field [111] differs from *parm94.dat* in torsion angle parameters involving the glycosidic torsion in nucleic acids. These serve to improve the predicted helical repeat and sugar pucker profiles.

3.9.3. The Wang et al. (1999) force field

`parm99.dat`	`Basic force field parameters`
`all_amino94.in`	`topologies and charges for amino acids`
`all_amino94nt.in`	`same, for N-terminal amino acids`
`all_amino94ct.in`	`same, for C-terminal amino acids`
`all_nuc94.in`	`topologies and charges for nucleic acids`
`gaff.dat`	`Force field for general organic molecules`
`all_modrna08.lib`	`topologies for modified nucleosides`
`all_modrna08.frcmod`	`parameters for modified nucleosides`

The **ff99** force field [112] points toward a common force field for proteins for "general" organic and bio-organic systems. The atom types are mostly those of Cornell *et al.* (see below), but changes have been made in many torsional parameters. The topology and coordinate files for the small molecule test cases used in the development of this force field are in the *parm99_lib* subdirectory. The *ff99* force field uses these parameters, along with the topologies and charges from the Cornell *et al.* force field, to create an all-atom nonpolarizable force field for proteins and nucleic acids.

There are more than 99 naturally occurring modifications in RNA. Amber force field parameters for all these modifications have been developed to be consistent with *ff94* and *ff99*.[44] The modular nature of RNA was taken into consideration in computing the atom-centered partial charges for these modified nucleosides, based on the charging model for the "normal" nucleotides.[113] All the *ab initio* calculations were done at the Hartree-Fock level of theory with 6-31G(d) basis sets, using the GAUSSIAN suite of programs. The computed electrostatic potential (ESP) was fit using RESP charge fitting in *antechamber*. Three-letter codes for all of the fitted nucleosides were developed to standardize the naming of the modified nucleosides in PDB files. For a detailed description of charge fitting for these nucleosides and an outline for the three letter codes, please refer to Ref. [44].

The AMBER force field parameters for 99 modified nucleosides are distributed in the form of library files. The *all_modrna08.lib* file contains coordinates, connectivity, and charges, and *all_modrna08.frcmod* contains information about bond lengths, angles, dihedrals and others. The AMBER force field parameters for the 99 modified nucleosides in RNA are also maintained at the modified RNA database at *http://ozone3.chem.wayne.edu*.

3.9.4. The 2002 polarizable force fields

```
frcmod.ff02pol.r1   Recommended initialization file
parm99.dat          Force field, for amino acids and some organic molecules;
                    can be used with either additive or
                    non-additive treatment of electrostatics.
parm99EP.dat        Like parm99.dat, but with "extra-points": off-center
                    atomic charges, somewhat like lone-pairs.
frcmod.ff02pol.r1   Updated torsion parameters for ff02.
all_nuc02.in        Nucleic acid input for building database, for a non-
                    additive (polarizable) force field without extra points.
all_amino02.in      Amino acid input ...
all_aminoct02.in    COO- amino acid input ...
all_aminont02.in    NH3+ amino acid input ....
all_nuc02EP.in      Nucleic acid input for building database, for a non-
                    additive (polarizable) force field with extra points.
all_amino02EP.in    Amino acid input ...
all_aminoct02EP.in  COO- amino acid input ...
all_aminont02EP.in  NH3+ amino acid input ....
```

The **ff02** force field is a polarizable variant of *ff99*. (See Ref. [114] for a recent overview of polarizable force fields.) Here, the charges were determined at the B3LYP/cc-pVTZ//HF/6-31G* level, and hence are more like "gas-phase" charges. During charge fitting the correction for intramolecular self polarization has been included.[86] Bond polarization arising from interactions with a condensed phase environment are achieved through polarizable dipoles attached to the atoms. These are determined from isotropic atomic polarizabilities assigned to each atom, taken from experimental work of Applequist. The dipoles can either be determined at each step through an iterative scheme, or can be treated as additional dynamical variables, and propagated through dynamics along with the atomic positions, in a manner analogous to Car-Parinello dynamics. Derivation of the polarizable force field required only minor changes in dihedral terms and a few modification of the van der Waals parameters.

Subsequently, a set up updated torsion parameters has been developed for the *ff02* polarizable force field.[115] These are available in the *frcmod.ff02pol.r1* file.

The user also has a choice to use the polarizable force field with extra points on which additional point charges are located; this is called **ff02EP**. The additional points are located on electron donating atoms (e.g. O,N,S), which mimic the presence of electron lone pairs.[116] For nucleic acids we chose to use extra interacting points only on nucleic acid bases and not on sugars or phosphate groups.

There is not (yet) a full published description of this, but a good deal of preliminary work on small molecules is available.[86, 117] Beyond small molecules, our initial tests have focused on small proteins and double helical oligonucleotides, in additive TIP3P water solution. Such a simulation model, (using a polarizable solute in a non-polarizable solvent) gains some of the advantages of polarization at only a small extra cost, compared to a standard force field model. In particular, the polarizable force field appears better suited to reproduce intermolecular interactions and directionality of H-bonding in biological systems than the additive force field. Initial tests show *ff02EP* behaves slightly better than *ff02*, but it is not yet clear how significant or widespread these differences will be.

3.9.5. Older ion parameters

In the past, for alkali ions with TIP3P waters, Amber has provided the values of Aqvist,[118] adjusted for Amber's nonbonded atom pair combining rules to give the same ion-OW potentials as in the original (which were designed for SPC water); these values reproduce the first peak of the radial distribution for ion-OW and the relative free energies of solvation in water of the various ions. Note that these values would have to be changed if a water model other than TIP3P were to be used. Rather arbitrarily, Amber also included chloride parameters from Dang.[119] These are now known not to work all that well with the Aqvist cation parameters, particularly for the K/Cl pair. Specifically, at concentrations above 200 mM, KCl will spontaneously crystallize; this is also seen with NaCl at concentrations above 1 M.[120] These "older" parameters are now collected in *frcmod.ionsff99_tip3p*, but are not recommended except to reproduce older simulations.

4. The Generalized Born/Surface Area Model

Implicit solvent methods can speed up atomistic simulations by approximating the discrete solvent as a continuum, thus drastically reducing the number of particles to keep track of in the system. An additional effective speedup often comes from much faster sampling of the conformational space afforded by these methods.[121–125] The generalized Born (GB) solvation model is the most commonly used implicit solvent model for atomistic MD simulation; it has been most widely tested on ff99SB and ff14SBonlysc, but in principle could be used with other non-polarizable force fields, such as ff03. To estimate the total solvation free energy of a molecule, ΔG_{solv}, one typically assumes that it can be decomposed into the "electrostatic" and "non-electrostatic" parts:

$$\Delta G_{solv} = \Delta G_{el} + \Delta G_{nonel} \tag{4.1}$$

where ΔG_{nonel} is the free energy of solvating a molecule from which all charges have been removed (i.e. partial charges of every atom are set to zero), and ΔG_{el} is the free energy of first removing all charges in the vacuum, and then adding them back in the presence of a continuum solvent environment. Generally speaking, ΔG_{nonel} comes from the combined effect of two types of interaction: the favorable van der Waals attraction between the solute and solvent molecules, and the unfavorable cost of breaking the structure of the solvent (water) around the solute. In the current Amber codes, this is taken to be proportional to the total solvent accessible surface area (SA) of the molecule, with a proportionality constant derived from experimental solvation energies of small non-polar molecules, and uses a fast LCPO algorithm [126] to compute an analytical approximation to the solvent accessible area of the molecule.

The Poisson-Boltzmann approach described in the next section has traditionally been used in calculating ΔG_{el}. However, in molecular dynamics applications, the associated computational costs are often very high, as the Poisson-Boltzmann equation needs to be solved every time the conformation of the molecule changes. Amber developers have pursued an alternative approach, the analytic generalized Born (GB) method, to obtain a reasonable, computationally efficient estimate to be used in molecular dynamics simulations. The methodology has become popular,[127–134] especially in molecular dynamics applications,[135–138] due to its relative simplicity and computational efficiency, compared to the more standard numerical solution of the Poisson-Boltzmann equation. Within Amber GB models, each atom in a molecule is represented as a sphere of radius R_i with a charge q_i at its center; the interior of the atom is assumed to be filled uniformly with a material of dielectric constant 1. The molecule is surrounded by a solvent of a high dielectric ε (80 for water at 300 K). The GB model approximates ΔG_{el} by an analytical formula,[127, 139]

$$\Delta G_{el} \approx -\frac{1}{2}\sum_{ij}\frac{q_i q_j}{f_{GB}(r_{ij},R_i,R_j)}\left(1-\frac{\exp[-\kappa f_{GB}]}{\varepsilon}\right) \tag{4.2}$$

where $r_{ij}$is the distance between atoms i and j , the R_i are the so-called *effective Born radii,* and $f_{GB}()$ is a certain smooth function of its arguments. The electrostatic screening effects of (monovalent) salt are incorporated [139] via the Debye-Huckel screening parameter κ.

A common choice [127] of f_{GB} is

$$f_{GB} = \left[r_{ij}^2 + R_i R_j \exp(-r_{ij}^2/4R_i R_j)\right]^{1/2} \tag{4.3}$$

although other expressions have been tried.[130, 140] The effective Born radius of an atom reflects the degree of its burial inside the molecule: for an isolated ion, it is equal to its van der Waals (VDW) radius ρ_i. Then one obtains the particularly simple form:

$$\Delta G_{el} = -\frac{q_i^2}{2\rho_i}\left(1-\frac{1}{\varepsilon}\right) \tag{4.4}$$

where we assumed $\kappa = 0$ (pure water). This is the famous expression due to Born for the solvation energy of a single ion. The function $f_{GB}()$ is designed to interpolate, in a clever manner, between the limit $r_{ij} \to 0$, when atomic spheres merge into one, and the opposite extreme $r_{ij} \to \infty$, when the ions can be treated as point charges obeying the Coulomb's law.[133] For deeply buried atoms, the effective radii are large, $R_i \gg \rho_i$, and for such atoms one can use a rough estimate $R_i \approx L_i$, where L_i is the distance from the atom to the molecular surface. Closer to the surface, the effective radii become smaller, and for a completely solvent exposed side-chain one can expect R_i to approach ρ_i.

The effective radii depend on the molecule's conformation, and so have to be re-computed every time the conformation changes. This makes the computational efficiency a critical issue, and various approximations are normally made that facilitate an effective estimate of R_i. With an exception of GBNSR6 (see Section 5.1), the so-called *Coulomb field approximation*, or *CFA*, is used for Amber GB models, which replaces the true electric displacement around the atom by the Coulomb field. Within this assumption, the following expression can be derived:[133]

$$R_i^{-1} = \rho_i^{-1} - \frac{1}{4\pi} \int \theta(|\mathbf{r}| - \rho_i) r^{-4} d^3\mathbf{r} \tag{4.5}$$

where the integral is over the solute volume surrounding atom i. For a realistic molecule, the solute boundary (molecular surface) is anything but trivial, and so further approximations are made to obtain a closed-form analytical expression for the above equation, *e.g.* the so-called pairwise de-screening approach of Hawkins, Cramer and Truhlar,[141] which leads to a GB model implemented in Amber with *igb=1*. The 3D integral used in the estimation of the effective radii is performed over the van der Waals (VDW) spheres of solute atoms, which implies a definition of the solute volume in terms of a set of spheres, rather than the complex molecular surface,[142] commonly used in the PB calculations. For macromolecules, this approach tends to underestimate the effective radii for buried atoms,[133] arguably because the standard integration procedure treats the small vacuum–filled crevices between the van der Waals (VDW) spheres of protein atoms as being filled with water, even for structures with large interior.[140] This error is expected to be greatest for deeply buried atoms characterized by large effective radii, while for the surface atoms it is largely canceled by the opposing error arising from the Coulomb approximation, which tends [128, 132, 143] to overestimate R_i.

The deficiency of the model described above can, to some extent, be corrected by noticing that even the optimal packing of hard spheres, which is a reasonable assumption for biomolecules, still occupies only about three quarters of the space, and so "scaling-up" of the integral by a factor of four thirds should effectively increase the underestimated radii by about the right amount, without any loss of computational efficiency. This idea was developed and applied in the context of pH titration,[133] where it was shown to improve the performance of the GB approximation in calculating pKa values of protein sidechains. However, the one-parameter correction introduced in Ref. [133] was not optimal in keeping the model's established performance on small molecules. It was therefore proposed [138] to re-scale the effective radii with the re-scaling parameters being proportional to the degree of the atom's burial, as quantified by the value I_i of the 3D integral. The latter is large for the deeply buried atoms and small for exposed ones. Consequently, one seeks a well-behaved re-scaling function, such that $R_i \approx (\rho_i^{-1} - I_i)^{-1}$ for small I_i, and $R_i > (\rho_i^{-1} - I_i)^{-1}$ when I_i becomes large. The following simple, infinitely differentiable re-scaling function was chosen to replace the model's original expression for the effective radii:

$$R_i^{-1} = \tilde{\rho}_i^{-1} - \rho_i^{-1} \tanh(\alpha\Psi - \beta\Psi^2 + \gamma\Psi^3) \tag{4.6}$$

where $\Psi = I_i\tilde{\rho}_i$, and α, β, γ are treated as adjustable dimensionless parameters which were optimized using the guidelines mentioned earlier (primarily agreement with the PB). Currently, Amber supports two GB models (termed OBC) based on this idea. These differ by the values of α, β, γ, and are invoked by setting igb to either igb=2 or igb=5. The details of the optimization procedure and the performance of the OBC model relative to the PB treatment and in MD simulations on proteins is described in Ref. [138]; an independent comparison to the PB in calculating the electrostatic part of solvation free energy on a large data set of proteins can be found in Ref. [144].

Our experience with generalized Born simulations is mainly with *ff99SB, ff14SBonlysc* or *ff03*; the current GB models are not compatible with polarizable force fields. Replacing explicit water with a GB model is equivalent to specifying a different force field, and users should be aware that none of the GB options (in Amber or elsewhere) is as mature as simulations with explicit solvent; user discretion is advised. For example, it was shown that

1	2	5	7	8
mbondi	*mbondi2*	*mbondi2*	*bondi*	*mbondi3*

Table 4.1.: *Recommended radii sets for various GB models. For values of igb given in the top row, the string in the second row should be entered in LEaP as "set default PBRadii xxx".*

salt bridges are too strong in some of these models [145, 146] and some of them provide secondary structure distributions that differ significantly from those obtained using the same protein parameters in explicit solvent, with GB having too much α-helix present.[147, 148] The combination of the *ff14SBonlysc* force field with igb=8 gives the best results for proteins [20][149], nucleic acids and protein-nucleic acid complexes. [150]

Despite these limitations, implicit treatment of solvent is widely used in molecular simulations for two main reasons: algorithmic/computational speed and conformational sampling. [125, 151] Implicit solvent methods can be algorithmically/computationally faster, as measured by simulation time steps per processor (CPU) time, because the vast number of individual interactions between the atoms of individual solvent molecules do not need to be explicitly computed. Implicit-solvent simulations can also sample conformational space faster in the low viscosity regime afforded by the implicit solvent model.[121–125] To some extent, the interest in implicit-solvent-based simulations is motivated by the need to sample very large conformational spaces for problems such as protein folding, binding-affinity calculations, or large-scale fluctuations of nucleosomal DNA fragments. The speedup of conformational change can vary considerably, depending on the details of the transition, and can range from no speedup at all to almost a 100-fold speedup. [125] In general, the larger the conformational change, the higher the speedup one may expect, but this tendency is not universal or uniform. These speedup values are also expected to vary by the specific flavour of GB model used, a detailed analysis for igb5 can be found in Ref. [125].

The generalized Born models used here are based on the "pairwise" model introduced by Hawkins, Cramer and Truhlar,[141, 152] which in turn is based on earlier ideas by Still and others.[127, 132, 143, 153] The so-called overlap parameters for most models are taken from the TINKER molecular modeling package (http://tinker.wustl.edu). The effects of added monovalent salt are included at a level that approximates the solutions of the linearized Poisson-Boltzmann equation.[139] The original implementation was by David Case, who thanks Charlie Brooks for inspiration. Details of our implementation of generalized Born models can be found in Refs. [154, 155].

4.1. GB/SA input parameters

As outlined above, there are several "flavors" of GB available, depending upon the value of *igb*. The version that has been most extensively tested corresponds to *igb*=1; the "OBC" models (*igb*=2 and 5) are newer, but appear to give significant improvements and are recommended for most projects (certainly for peptides or proteins). The newest, most advanced, and least extensively tested model, *GBn* (*igb*=7), yields results in considerably better agreement with molecular surface Poisson-Boltzmann and explicit solvent results than the "OBC" models under many circumstances.[148] The *GBn* model was parameterized for peptide and protein systems and is not recommended for use with nucleic acids. A modification on the GBn model (igb=8) further improves agreement between Poisson-Boltzmann and explicit solvent data compared to the original formulation (igb=7).[20] Users should understand that all (current) GB models have limitations and should proceed with caution. Generalized Born simulations can only be run for non-periodic systems, *i.e.* where *ntb*=0. Unlike its use in explicit solvent PME simulations, short nonbonded cutoff values have much stronger impact on accuracy of the GB calculations. Essentially, any cutoff values other than *cut* > *structure size* can lead to artifacts. Current GPU implementation of the GB can not use cutoffs. An alternative that retains most of the speed of the GB with a cutoff, but without most of its artifacts, is GB-HCP described in Section 39.6. If the nonbonded cutoff is used in GB calculations, it should be greater than that for PME calculations, perhaps cut=16. The slowly-varying forces generally do not have to be evaluated at every step for GB, either nrespa=2 or 4, although that option may lead to some artifacts as well.

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

= 1 The Hawkins, Cramer, Truhlar[141, 152] pairwise generalized Born model is used, with parameters described by Tsui and Case.[154] This model uses the default radii set up by LEaP. It is slightly different from the GB model that was included in Amber6. If you want to compare to Amber 6, or need to continue an ongoing simulation, you should use the command "set default PBradii amber6" in LEaP, and set *igb*=1 in *sander*. For reference, the Amber6 values are those used by an earlier Tsui and Case paper.[136] Note that most nucleic acid simulations have used this model, so you take care when using other values. Also note that Tsui and Case used an offset (see below) of 0.13 Å, which is different from its default value.

= 2 Use a modified GB model developed by A. Onufriev, D. Bashford and D.A. Case; the main idea was published earlier,[133] but the actual implementation here[138] is an elaboration of this initial idea. Within this model, the effective Born radii are re-scaled to account for the interstitial spaces between atom spheres missed by the GB^{HCT} approximation. In that sense, GB^{OBC} is intended to be a closer approximation to true molecular volume, albeit in an average sense. With *igb*=2, the inverse of the effective Born radius is given

by:cedure

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

where $\overline{\rho}_i = \rho_i - offset$, and $\Psi = I\rho_i$, with I given in our earlier paper. The parameters α, β, and γ were determined by empirical fits, and have the values 0.8, 0.0, and 2.909125. This corresponds to model I in Ref [138]. With this option, you should use the LEaP command "set default PBradii mbondi2" to prepare the *prmtop* file.

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

= 5 Same as *igb=2*, except that now α, β, γ are 1.0, 0.8, and 4.85. This corresponds to model II in Ref [138]. With this option, you should use the command "set default PBradii mbondi2" in setting up the *prmtop* file, although "set default PBradii bondi" is also OK. When tested in MD simulations of several proteins,[138] both of the above parameterizations of the "OBC" model showed equal performance, although further tests [144] on an extensive set of protein structures revealed that the *igb=5* variant agrees better with the Poisson-Boltzmann treatment in calculating the electrostatic part of the solvation free energy.

= 6 With this option, there is no continuum solvent model used at all; this corresponds to a non-periodic, "vacuum", model where the non-bonded interactions are just Lennard-Jones and Coulomb interactions. This option is logically equivalent to setting *igb=0* and *eedmeth=4*, although the implementation (and computational efficiency) is not the same.

= 7 The *GBn* model described by Mongan, Simmerling, McCammon, Case and Onufriev[156] is employed. This model uses a pairwise correction term to GB^{HCT} to approximate a molecular surface dielectric boundary; that is to eliminate interstitial regions of high dielectric smaller than a solvent molecule. This correction affects all atoms and is geometry-specific, going beyond the geometry-free, "average" re-scaling approach of GB^{OBC}, which mostly affects buried atoms. With this method, you should use the bondi radii set. The overlap or screening parameters in the *prmtop* file are ignored, and the model-specific *GBn* optimized values are substituted. The model carries little additional computational overhead relative to the other GB models described above.[156] This method is not recommended fcedureor systems involving nucleic acids.

= 8 Same GB functional form as the GBn model (igb=7), but with different parameters. The offset, overlap screening parameters, and gbneckscale are changed. In addition, individual α , β , and γ parameters can be specified for each of the elements H, C, N, O, S, P. Parameters for other elements have not been optimized, and the default values used are the ones from igb=5, which were not element-dependent. Default values were optimized for H, C, N, O and S atoms in protein systems.[20] Although the parameters for P in proteins can be specified, the default values were not optimized and are the igb=5 values. Nucleic acids have separate parameters from those used for proteins, and default values were optimized for H, C, N, O and P atoms in nucleic acid systems.[150]

The following are the default parameters sander uses with *igb=8*:

```
Sh=1.425952, Sc=1.058554, Sn=0.733599,
So=1.061039, Ss=-0.703469, Sp=0.5,
offset=0.195141, gbneckscale=0.826836,
gbalphaH=0.788440, gbbetaH=0.798699, gbgammaH=0.437334,
gbalphaC=0.733756, gbbetaC=0.506378, gbgammaC=0.205844,
gbalphaN=0.503364, gbbetaN=0.316828, gbgammaN=0.192915,
gbalphaOS=0.867814, gbbetaOS=0.876635, gbgammaOS=0.387882,
gbalphaP=1.0, gbbetaP=0.8, gbgammaP=4.85
screen_hnu=1.69654, screen_cnu=1.26890,
screen_nnu=1.425974, screen_onu=0.18401, screen_pnu=1.54506,
gb_alpha_hnu=0.53705, gb_beta_hnu=0.36286, gb_gamma_hnu=0.11670,
gb_alpha_cnu=0.33167, gb_beta_cnu=0.19684, gb_gamma_cnu=0.09342,
gb_alpha_nnu=0.68631, gb_beta_nnu=0.46319, gb_gamma_nnu=0.13872,
gb_alpha_onu=0.60634, gb_beta_onu=0.46301, gb_gamma_onu=0.14226,
gb_alpha_pnu=0.41836, gb_beta_pnu=0.29005, gb_gamma_pnu=0.10642
```

Parameters for proteins and for nucleic acids were optimized separately and can be independently specified. Protein parameters: Sh, Sc, Sn, So, Ss and Sp are scaling parameters, gbalphaX, gbbetaX, gbgammaX are the α , β , γ set for element X. gbalphaOS, gbbetaOS, gbgammaOS is the α , β , γ set applied to both O and S. The phosphorus parameters (in proteins) were not optimized and are simply taken as the parameters used in the OBC-2 model (igb=5). Nucleic acid parameters (end with "nu"): screen_Xnu (X=h, c, n, o, p) are scaling parameters, gb_alpha_Xnu (X=h, c, n, o, p) are the α , β , γ set for element X.

Since parameters are assigned for each atom based on its residue name (hard-coded in "sander/egb.F90" (subroutine isnucat)), users need to update the residue table in the sander source code if nucleic acids with different names are simulated using this GB model.

The default values for offset=0.195141, gbneckscale=0.826836 are recommended for both proteins and nucleic acids.

mbondi3 radii are recommended with igb=8 and can be employed with the LEaP command "set default PBradii mbondi3". The mbondi3 radii were adjusted based on protein simulations, and optimization of these radii for nucleic acids is currently underway.

=10 Calculate the reaction field and nonbonded interactions using a numerical Poisson-Boltzmann solver. This option is described in the Chapter 6. Note that this is *not* a generalized Born simulation, in spite of its use of *igb*; it is rather an alternative continuum solvent model.

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

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

saltcon Sets the concentration (M) of 1-1 mobile counterions in solution, using a modified generalized Born theory based on the Debye-Hückel limiting law for ion screening of interactions.[139] Default is 0.0 M (*i.e.* no Debye-Hückel screening.) Setting *saltcon* to a nonzero value does result in some increase in computation time.

rgbmax This parameter controls the maximum distance between atom pairs that will be considered in carrying out the pairwise summation involved in calculating the effective Born radii. Atoms whose associated spheres are farther way than *rgbmax* from given atom will not contribute to that atom's effective Born radius. This is implemented in a "smooth" fashion (thanks mainly to W.A. Svrcek-Seiler), so that when part of an atom's atomic sphere lies inside *rgbmax* cutoff, that part contributes to the low-dielectric region that determines the effective Born radius. The default is 25 Å, which is usually plenty for single-domain proteins of a few hundred residues. Even smaller values (of 10-15 Å) are reasonable, changing the functional form of the generalized Born theory a little bit, in exchange for a considerable speed-up in efficiency, and without introducing the usual cut-off artifacts such as drifts in the total

energy.
The *rgbmax* parameter affects only the effective Born radii (and the derivatives of these values with respect to atomic coordinates). The *cut* parameter, on the other hand, determines the maximum distance for the electrostatic, van der Waals and "off-diagonal" terms of the generalized Born interaction. The value of *rgbmax* might be either greater or smaller than that of *cut*: these two parameters are independent of each other. However, values of *cut* that are too small are more likely to lead to artifacts than are small values of *rgbmax*; therefore one typically sets *rgbmax* <= *cut*.

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

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

gbsa Option to carry out GB/SA (generalized Born/surface area) simulations. For the default value of 0, surface area will not be computed and will not be included in the solvation term. If *gbsa = 1*, surface area will be computed using the LCPO model.[126] If *gbsa = 2*, surface area will be computed by recursively approximating a sphere around an atom, starting from an icosahedra. Note that no forces are generated in this case, hence, *gbsa = 2* only works for a single point energy calculation and is mainly intended for energy decomposition in the realm of MM-GBSA.

surften Surface tension used to calculate the nonpolar contribution to the free energy of solvation (when *gbsa = 1*), as Enp = surften*SA. The default is 0.005 kcal/mol/A^2.[157]

rdt This parameter is only used for GB simulations with LES (Locally Enhanced Sampling). In GB+LES simulations, non-LES atoms require multiple effective Born radii due to alternate descreening effects of different LES copies. When the multiple radii for a non-LES atom differ by less than RDT, only a single radius will be used for that atom. See Chapter 24 for more details. Default is 0.0 Å.

4.2. ALPB (Analytical Linearized Poisson-Boltzmann)

Like the GB model, the ALPB approximation [158, 159] can be used to replace the need for explicit solvent, with similar benefits (such as enhanced conformational sampling) and caveats. The basic ALPB equation that approximates the electrostatic part of the solvation free energy is

$$\Delta G_{el} \approx \Delta G_{alpb} = -\frac{1}{2}\left(\frac{1}{\varepsilon_{in}} - \frac{1}{\varepsilon_{ex}}\right)\frac{1}{1+\alpha\beta}\sum_{ij} q_i q_j \left(\frac{1}{f_{GB}} + \frac{\alpha\beta}{A}\right) \tag{4.7}$$

where $\beta = \varepsilon_{in}/\varepsilon_{ex}$ is the ratio of the internal and external dielectrics, α=0.571412, and A is the so-called *effective electrostatic size* of the molecule, see the definition of *Arad* below. Here f_{GB} is the same smooth function as in the GB model. The GB approximation is then just the special case of the ALPB when the solvent dielectric is infinite; however, for finite values of solvent dielectric the ALPB tends to be more accurate. For aqueous solvation, the accuracy advantage offered by the ALPB is still noticeable, and becomes more pronounced for less polar solvents. Statistically significant tests on macromolecular structures [159] have shown that ALPB is more likely to be a better approximation to PB than the GB. At the same time, the ALPB has virtually no additional computational overhead relative to GB. However, users should realize that at this point the new model has not yet been tested nearly as extensively as the canonocal GB model. The ALPB can potentially replace the GB in the energy analysis of snapshots via the MM-GB/SA scheme. The electrostatic screening effects of monovalent salt are currently introduced into the ALPB in the same manner as in the GB, and are determined by the parameter *saltcon* .

alpb Flag for using ALPB to handle electrostatic interactions within the implicit solvent model.

= 0 No ALPB (default).

= 1 ALPB is turned on. Requires that one of the analytical GB models is also used to compute the effective Born radii, that is one must set *igb*=1,2,5, or 7. The ALPB uses the same sets of radii as required by the particular GB model.

arad	Effective electrostatic size (radius) of the molecule. Characterizes its over-all dimensions and global shape, and is not to be confused with the effective Born radius of an atom. An appropriate value of *Arad* must be set if *alpb=1*: this can be conveniently estimated for your input structure with the utility *elsize* that comes with the main distribution. The default is 15 Å. While *Arad* may change during the course of a simulation, these changes are usually not very large; the accuracy of the ALPB is found to be rather insensitive to these variations. In the current version of Amber *Arad* is treated as constant throughout the simulation, the validity of this assumption is discussed in Ref. [159]. Currently, the effective electrostatic size is only defined for "single-connected" molecules. However, the ALPB model can still be used to treat the important case of complex formation. In the docked state, the compound is considered as one, with its electrostatic size well defined. When the ligand and receptor become infinitely separated, each can be assigned its own value of *Arad*.

4.2.1. elsize

NAME

```
elsize - Given the structure, estimates its effective electrostatic size
    (parameter Arad ) need by the ALPB model.
```

SYNOPSIS

```
Usage: elsize input-pqr-file [-options]
-det an estimate based on structural invariants. DEFAULT.
-ell an estimate via elliptic integral (numerical).
-elf same as above, but via elementary functions.
-abc prints semi-axes of the effective ellipsoid.
-tab prints all of the above into a table without header.
-hea prints same table as -tab but with a header.
-deb prints same as -tab with some debugging information.
-xyz uses a file containing only XYZ coordinates.
```

DESCRIPTION

elsize is a program originally written by G. Sigalov to estimate the effective electrostatic size of a structure via a quick, analytical method. The algorithm is presented in detail in Ref. .[159] You will need your structure in a pqr format as input, which can be easily obtained from the prmtop and inpcrd files using *ambpdb* utility described above:

```
ambpdb -p prmtop -pqr -c inpcrd > input-file-pqr
```

After that you can simply do: *elsize input-file-pqr* , the value of electrostatic size in Angstroms will be output on stdout. The source code is in the src/etc/ directory, its comments contain more extensive description of the options and give an outline of the algorithm. A somewhat less accurate estimate uses just the XYZ coordinates of the molecule and assumes the default radius size of for all atoms:

```
elsize input-file-xyz
```

This option is not recommended for very small compounds. The code should not be used on structures made up of two or more completely disjoint" compounds – while the code will still produce a finite value of *Arad* , it is not very meaningful. Instead, one should obtain estimates for each compound separately.

5. GBNSR6

GBNSR6 is an implementation of the Generalized Born (GB) model in which the effective Born radii are computed numerically, via the so-called "R6" integration[160, 161] over molecular surface of the solute:

$$\mathbf{R_i}^{-1} = \left(-\frac{1}{4\pi} \oint_{\partial V} \frac{\mathbf{r} - \mathbf{r_i}}{|\mathbf{r}-\mathbf{r_i}|^6} \cdot \mathbf{dS} \right)^{1/3} \tag{5.1}$$

For most structures, GB solvation based on the numerical R6 radii are virtually as accurate[156] as the GB energies based on the "gold standard" perfect effective radii, which can in principle be obtained from numerical solution of the PB equation[140]. As a result, the numerical R6 formulation is generally more accurate than the fast analytical approaches described above. In contrast to most GB practical models, GBNSR6 model is parameter-free in the same sense as the numerical PB framework is. Thus, accuracy of GBNSR6 relative to the PB standard is virtually unaffected by the choice of input atomic radii. However, unlike the analytical GB models in AMBER, GBNSR6 can not yet be used in dynamics. Recent benchmarks show that electrostatic binding energies computed by GBNSR6 are in good agreement with the numerical PB reference[162].

Within GBNSR6, any of the following three versions of the pairwise GB equation can be used for computation of the solvation energies: (1) the canonical (Still 1990) GB[127], (2) the canonical GB with the ALPB correction[158, 159], and (3) the charge hydration asymmetric generalized Born (CHAGB) model[163]. The models are listed below; the first two are described in more detail in the GB section of the main manual, a brief introduction into CHAGB is below. For more information about these models please refer to the original references.

5.1. GB equations available in gbnsr6

- Canonical GB: the original equation due to Still et al, Eqs.4.2, 4.3.
- ALPB: an inexpensive correction, Eq. 4.7, to Still's equation that restores correct dependence on dielectric constants. The correction is recommended in all cases except small molecules with decidedly non-spherical topology (e.g. rings) or structures that are topologically not singly-connected, e.g. two molecules not in contact with each other. The electrostatic size is computed automatically, no need to specify it in GBNSR6.
- CHAGB: The effect of charge hydration asymmetry (CHA)[89] – non-invariance of solvation free energy upon solute charge inversion – is incorporated into the Generalized Born framework[163]. The CHA is added to the GB equation (with or without the ALPB correction) to emulate asymmetric response to solvated charge of the specified explicit water model, e.g. TIP3P; the asymmetric response, which can be very strong, is ultimately determined by the charge distribution within the water model. Note that in contrast to standard GB or PB, CHAGB employs a novel definition of the dielectric boundary that does not subsume the CHA effects into the intrinsic atomic radii, therefore a special input radii set is used with this model. This model has so far been tested on a diverse set of neutral small molecules, charged and uncharged amino acid analogs and small proteins. Noticeable accuracy improvement over the uncorrected GB was reported for individual solvation energies. The optimum radii set for CHAGB available in this implementation shows better transferability between different classes of molecules. However, the model has not been tested in the context of protein-ligand binding, which may require a different radii set for optimum performance.

5.2. Numerical implementation of the R6 integral

- The R6 integral for computing the effective Born radius, Eq. 5.1, is performed for each atom over grid-based molecular surface of the solute. The molecular surface is based on the field-view method[164] also used in the PBSA tool. A uniform Cartesian grid is utilized to discretize a rectangular box containing the molecular structure. By exploiting the conservation of "electric flux" through the surface, the resulting finite difference grid surface elements traverse the same solid angle as the spherical surface elements obtained from the Lee and Richards molecular surface. More details of this implementation can be found in Ref.[164].

5.3. Usage

Just like other GB models available in AMBER, GBNSR6 can be used for efficient estimates of solvation free energy in situation where numerical PB estimates are too expensive. In addition to the value of the total solvation free energy, ΔG,its pairwise decomposition ΔG_{ij} can be obtained without significant additional computational expense typically associated with such estimates within the PB formalism. Options to output components of the non-polar solvation energy are available as well.

5.3.1. Input files

gbnsr6 has a similar usage as amber/sander:
gbnsr6 -i mdin -o mdout -p prmtop -c inpcrd

mdin input control data for the computations.

mdout output of the program in a user readable state info and diagnostics. "-o stdout" will send the output to the terminal.

prmtop input molecular topology file.

inpcrd input initial coordinate file.

5.3.2. Basic input options

The input file is very similar to the Amber/sander format. There are two namelist &cntrl and &gb . The only flag available in &cntrl is inp, the rest of the flags are in the namelist &gb. The following is a description of the available flags:

B	Specifies the value of uniform offset [156] to the (inverse) effective radii, the default value is 0.028 Å^{-1} which gives better agreement with the PB model, regardless of the structure size. For best agreement with the explicit solvent (TIP3P) solvation energies, optimal value of B depends on the structure size: for small molecules (number of atoms less than 50), we recommend B=0. With -chagb option, B is calculated automatically based on the solute size.
alpb	Specifies if ALBP correction is to be used. **=** 0 Canonical GB is used. **=** 1 ALPB is used (default)
epsin	Sets the dielectric constant of the solute region, default is 1.0. The solute region is defined to be the solvent excluded volume.
epsout	Sets the implicit solvent dielectric constant for the solvent, the default value is 78.5.
istrng	Sets the ionic strength (in mM) for the GB equation. Default is 0 mM. Physiological monovalent salt would correspond to 145 mM. Note the unit is different from that (in M) used by the other generalized Born methods implemented in Amber.
Rs	Sets the value of the dielectric boundary shift compared to the molecular surface, default value is 0.52Å (only relevant for the -chagb option).
dprob	Sets the radius of the solvent robe, default is 1.4 Å.

space	Sets the grid spacing that determines the resolution of the solute molecular surface, default is 0.5 Å. Note that memory footprint of this grid-based implementation of GBNSR6 may become large for large structures, e.g. the nucleosome (about 25,000 atoms) will take close to 2 GB of RAM when the default grid spacing is used. For very large structures, one may consider increasing the value of space, which will reduce the memory footprint and execution time; however, the accuracy will also decrease.
arcres	Sets the arc resolution used for numerical integration over molecular surface, the default value is 0.2 Å.
rbornstat	= 0 values of the inverse effective Born radii are not printed (default). = 1 print the inverse effective Born radii to the outfile.
dgij	This flag is used for printing pairwise electrostatic energies. The values will be found in the output file, starting with the label "DGij". The second and third columns of these lines especify the atom indexes of the respective atomic pair. Energy units are kcal/mol. = 0 does not print pairwise terms (default). = 1 prints polar component only of the solvation energy between all pairs of atoms.
radiopt	Specifies the set of intrinsic atomic radii to be used with the chagb option. = 0 uses hardcoded intrisic radii optimized for small drug like molecules, and single amino acid dipeptides[163] (default) = 1 intrinsic radii are read from the topology file. Note that the dielectric surface defined using these radii is then shifted outwards by Rs relative to the molecular surface. The option is not recommended unless you are planning to re-optimize the input radii set for your problem.
chagb	= 0 Do not use CHAGB (default). = 1 Use CHAGB.
ROH	Sets the value of R^z_{OH} for CHA GB model, the default is 0.586Å. This parameter defines which explicit water model is being mimicked with respect to its propensity to cause CHA, the default corresponds to TIP3P and SPC/E. For OPC, $R^z_{OH} = 0.699$Å, for TIP4P $R^z_{OH} = 0.734$Å, and 0.183Å for TIP5P/E. A perfectly tetrahedral water , which can not cause charge hydration asymmetry, would have $R^z_{OH} = 0$.
tau	Sets the value of τ in the CHAGB model, the default is 1.47. This dimensionless parameter controls the effective range of the neighboring charges (j) affecting the CHA of atom (i), see Ref.[163] for details.
inp	= 0 do not compute nonpolar solvation energy. = 1 compute nonpolar solvation energies.
cavity_surften	Sets the surface tension parameter for nonpolar solvation calculation, the default value is 0.005 (kcal/mol/A^2). This will be read only if the inp=1.

More options are available in a stand-alone version of GBNSR6 code not based on Cartesian grid [160].

5.3.3. Examples of input files

Compute electrostatic energy using default parameters.

```
&cntrl
 inp=0
/
```

Compute electrostatic energies including nonpolar solvation energies and print the inverse effective Born radii

```
&cntrl
 np=1
/
&gb
 epsin=1.0, epsout=78.5, istrng=0, dprob=1.4, space=0.5,
 arcres=0.2, B=0.028, alpb=1, rbornstat=1, cavity_surften=0.005
/
```

Use chagb to compute solvation energy, include ALPB correction.

```
&cntrl
 inp=1
/
&gb
 alpb=1, chagb=1
/
```

6. PBSA

Several efficient finite-difference numerical solvers, both linear [165, 166] and nonlinear,[167] are implemented in *pbsa* for various applications of the Poisson-Boltzmann method. In the following, a brief introduction is given on the method, the numerical solvers, and numerical energy and force calculations. This is followed by a detailed description of the usage and keywords. Finally example input files are explained for typical *pbsa* applications. For more information on the background and how to use the method, please consult cited references and online *Amber* tutorial pages.

6.1. Introduction

Solvation interactions, especially solvent-mediated dielectric screening and Debye-Hückel screening, are essential determinants of the structure and function of proteins and nucleic acids.[168] Ideally, one would like to provide a detailed description of solvation through explicit simulation of a large number of solvent molecules and ions. This approach is frequently used in molecular dynamics simulations of solution systems. In many applications, however, the solute is the focus of interest, and the detailed properties of the solvent are not of central importance. In such cases, a simplified representation of solvation, based on an approximation of the mean-force potential for the solvation interactions, can be employed to accelerate the computation.

The mean-force potential averages out the degrees of freedom of the solvent molecules, so that they are often called implicit or continuum solvents. The formalism with which implicit solvents can be applied in molecular mechanics simulations is based on a rigorous foundation in statistical mechanics, at least for additive molecular mechanics force fields. Within the formalism, it is straightforward to understand how to decompose the total mean-field solvation interaction into electrostatic and non-electrostatic components that scale quite differently and must be modeled separately (see for example [169]).

The Poisson-Boltzmann (PB) solvents are a class of widely used implicit solvents to model solvent-mediated electrostatic interactions.[168] They have been demonstrated to be reliable in reproducing the energetics and conformations as compared with explicit solvent simulations and experimental measurements for a wide range of systems.[168] In these models, a solute is represented by an atomic-detail model as in a molecular mechanics force field, while the solvent molecules and any dissolved electrolyte are treated as a structure-less continuum. The continuum treatment represents the solute as a dielectric body whose shape is defined by atomic coordinates and atomic cavity radii.[170] The solute contains a set of point charges at atomic centers that produce an electrostatic field in the solute region and the solvent region. The electrostatic field in such a system, including the solvent reaction field and the Coulombic field, may be computed by solving the PB equation:[171, 172]

$$\nabla \cdot [\varepsilon(\mathbf{r})\nabla\phi(\mathbf{r})] = -4\pi\rho(\mathbf{r}) - 4\pi\lambda(\mathbf{r})\sum_i z_i c_i \exp(-z_i\phi(\mathbf{r})/k_B T) \tag{6.1}$$

where $\varepsilon(\mathbf{r})$ is the dielectric constant, $\phi(\mathbf{r})$ is the electrostatic potential, $\rho(\mathbf{r})$ is the solute charge, $\lambda(\mathbf{r})$ is the Stern layer masking function, z_i is the charge of ion type i, c_i is the bulk number density of ion type i far from the solute, k_B is the Boltzmann constant, and T is the temperature; the summation is over all different ion types. The salt term in the PB equation can be linearized when the Boltzmann factor is close to zero. However, the approximation apparently does not hold in highly charged systems. Thus, it is recommended that the full nonlinear PB equation solvers be used in such systems.

The non-electrostatic or non-polar (NP) solvation interactions are typically modeled with a term proportional to the solvent accessible surface area (SASA).[157] An alternative and more accurate method to model the non-polar solvation interactions is also implemented in *pbsa*.[173] The new method separates the non-polar solvation interactions into two terms: the attractive (dispersion) and repulsive (cavity) interactions. Doing so significantly improves the correlation between the cavity free energies and solvent accessible surface areas or molecular volumes enclosed by SASA for branched and cyclic organic molecules.[174] This is in contrast to the commonly used strategy that correlates total non-polar solvation energies with solvent accessible surface areas, which only correlates well for linear aliphatic molecules.[157] In the alternative method, the attractive free energy is computed by a numerical integration over the solvent accessible surface area that accounts for solvation attractive interactions with an infinite cutoff.[175]

6.1.1. Numerical solutions of the PB equation

In *pbsa* both the linear form and the full nonlinear form of the PB equation are supported. Many strategies may be used to discretize the PB equation, but only the finite-difference (FD) method, or more rigorously, the finite-volume method [176–178] is fully supported in *pbsa* for both the linear and nonlinear PB equations. A FD method involves the following steps: mapping atomic charges to the FD grid points (termed grid charges below); assigning non-periodic/periodic boundary conditions, *i.e.,* electrostatic potentials on the boundary surfaces of the FD grid; and applying a dielectric model to define the high-dielectric (i.e., water) and low-dielectric (i.e., solute interior) regions and mapping it to the FD grid edges.

These steps allow the partial differential equation to be converted into a linear or nonlinear system with the electrostatic potential on grid points as unknowns, the charge distribution on the grid points as the source, and the dielectric constant on the grid edges (and the salt-related term for the linear case) wrapped into the coefficient matrix, which is a seven-banded symmetric matrix. In *pbsa*, four common linear FD solvers are implemented: modified ICCG, geometric multigrid, conjugate gradient, and successive over-relaxation (SOR).[166] In addition, we have also implemented six nonlinear FD solvers: Inexact Newton(NT)/modified ICCG, NT/geometric multigrid, conjugate gradient, and SOR and its improved versions - adaptive SOR and damped SOR.[167]

In addition to the FD method, a new discretization strategy is also introduced to solve the linear PB equation.[179] The Immersed Interface method (IIM) is a second-order accurate numerical method developed for systems with interface, i.e. solute/solvent boundary in this case. In the IIM discretization scheme, the linear equations on regular grid points, i.e. grid points away from the interface, are the same as the standard finite-difference method, but the linear equations on irregular grid points, i.e. grid points nearby the interface, are constructed by minimizing the magnitude of the local truncation error in the discretization of the PB equation.[180] It can be proven that the errors of calculated potentials are at the order of $O(h^2)$ on the regular grid points and $O(h)$ on the irregular grid points.[180]

6.1.2. Numerical interpretation of energy and forces

PB solvents approximate the solvent-induced electrostatic mean-force potential by computing the reversible work in the process of charging the atomic charges in a solute molecule or complex. The charging free energy is a function of the electrostatic potential ϕ, which can be computed by solving the linear or nonlinear system.

It has been shown (see for example [169]) that the total electrostatic energy of a solute molecule can be approximated through the FD approach by subtracting the self FD Coulombic energy ($G^{FD}_{coul,shelf}$) and the short-range FD Coulombic energy ($G^{FD}_{coul,short}$) from the total FD electrostatic energy ($G^{FD}_{coul,total}$), and adding back the analytical short-range Coulombic energy ($G^{ana}_{coul,short}$). The self FD Coulombic energy is due to interactions of grid charges within one single atom. The self energy exists even when the atomic charge is exactly positioned on one grid point. It also exists in the absence of solvent and any other charges. It apparently is a pure artifact of the FD approach and must be removed. The short-range FD Coulombic energy is due to interactions between grid charges in two different atoms that are separated by a short distance, usually less than 14 grid units. The short-range Coulombic energy is inaccurate because the atomic charges are mapped onto their eight nearest FD grids, thus causing deviation from the analytical Coulomb energy. The correction of $G^{FD}_{coul,shelf}$ and $G^{FD}_{coul,short}$is made possible by the work of Luty and McCammon's analytical approach to compute FD Coulombic interactions.[181]

Therefore, the PB electrostatic interactions include both Coulombic interactions and reaction field interactions for all atoms of the solute. The total electrostatic energy is given in the energy component EEL in the output file. The term that is reserved for the reaction field energy, EPB, is zero if this method is used. If you want to know how much of EEL is the reaction field energy, you can set the BCOPT keyword (to be explained below) to compute the reaction field energy only by using a Coulombic field (or singularity) free formulation.[182]

When the full nonlinear Poisson-Boltzmann equation is used, an additional energy term, the ionic energy, should also be included. This energy term disappears in the symmetrical linear system because the effects due to opposite ions cancel out. It is currently approximated by calculation up to the space boundary of the FD grid. It should be noted that the NBUFFER keyword may need increasing to obtain good precision in the ionic energy for small molecules with a large FILLRATIO.

An alternative method of computing the electrostatic interactions is also implemented in *pbsa*. In this method, the reaction field energy is computed directly after the induced surface charges are first computed at the dielectric boundary (i.e., the surface that separates solute and solvent). These surface charges are then used to compute the reaction field energy,[168] and is given as the EPB term. It has been shown that doing so improves the convergence of reaction field energy with respect to the FD grid spacing. However, a limitation of this method is that the Coulombic energy has to be recomputed analytically with a pairwise summation procedure. When this method is used, the EEL term only gives the Coulombic energy with a cutoff distance provided in the input file. The two ways of computing electrostatic interactions are controlled by the keywords ENEOPT and FRCOPT to be described below.

The non-polar solvation free energy is returned by the ECAVITY term, which is either the total non-polar solvation free energy or the cavity solvation free energy in the two different models described above. The EDISPER term returns the dispersion solvation free energy. Of course it is zero if the total non-polar solvation free energy has been returned by ECAVITY. The word INP can be used to choose one of the two treatments of non-polar solvation interactions.[173] Specifically, you can use SASA to correlate total non-polar solvation free energy, i.e., $G_{np} = NP\_TENSION \times SASA + NP\_OFFSET$ as in PARSE.[157] You can also use SASA to correlate the cavity term only and use a surface-integration approach to compute the dispersion term.[173] i.e., G_{np}= G_{disp}+ G_{cavity}, with $G_{cavity} = CAVITY\_TENSION \times SASA + CAVITY\_OFFSET$. See the discussion of keywords in 8.2.8. These options are described in detail in Ref. [173].

Finally, in this release, the PB forces are now correctly interpreted for the widely used SES molecular surface definition, *i.e.,* the partition of dielectric boundary pressure/force can now reproduce the virtual work principle. This is achieved by proper decomposition of the dielectric boundary force on the reentrant portion of the molecular surface. Specifically, the molecular surface is computed more accurately by considering the cases when the solvent probe touches three atoms simultaneously. Next the reentrant force is also distributed onto the three atoms forming the reentrant surface following the virtual work principle.[183]

6.1.3. Numerical accuracy and related issues

Note that the accuracy of any numerical PB procedure is determined by the discretization resolution specified in the input, *i.e.,* the grid spacing. The convergence criterion for the iteration procedures also plays some role for the numerical PB solvers. Finally the accuracy is highly dependent upon the methods used for computing total electrostatic interactions. In Lu and Luo,[169] the accuracy of the first method for total electrostatic interactions is discussed in detail. In Ref.[183] the accuracy of the second method is discussed.

It is recommended that the second method for total electrostatic interactions be used for most calculations. Apparently the cutoff distance for charge-charge interactions strongly influences the accuracy of electrostatic interactions. The default setting is infinity, i.e., no cutoff is used. In this method, the convergence of the reaction field energy with respect to the grid spacing is much better than that of the first method. Our experience shows that the reaction field energies converge to within ~2% for tested proteins at the grid spacing of 0.5 Å when the weighted harmonic average of dielectric constants is used at the solute/solvent interface (when SMOOTHOPT = 1, see below).[184]

The reaction field energies computed with the second method (when SMOOTHOPT = 2) are also in excellent agreement (differences in the order of 0.1%) with those computed with the *Delphi* program which uses the same method for energy calculation. For example, see the computational set up documented in test case *pbsa_delphi* in this release.[185]

The accuracy of non-polar solvation energy depends on the quality of SASA which is computed numerically by representing each atomic surface by spherically distributed dots. Thus a higher dot density gives more accurate atomic surface and molecular surface. However, it is found that the default setting for the dot density is quite sufficient for typical applications.[173] Should you encounter any memory allocation error for surface calculation, you are advised to use a coarser surface dot resolution if the physical memory of your computer is limited.

Numerical solvation calculations are memory intensive for macromolecules due to the fine grid resolution required for sufficient accuracy. Thus, the efficiency of *pbsa* depends on how much memory is allocated for it and the performance of the memory subsystem. The option that is directly related to its memory allocation is the FD grid spacing for the PB equation and the surface dot resolution for molecular surface. Apparently the geometric dimension and the number of atoms are also important for predicting the memory usage. In general for a typical computer configuration with 8GB memory, the geometric dimension can be as large as $180 \times 180 \times 180$ Å^3 at the default grid spacing of 0.5 Å before the computer responds too slowly.

6.2. Usage and keywords

6.2.1. File usage

pbsa has a very similar user interface as the *Amber/sander* program, though much simpler.

```
pbsa [-O] -i mdin -o mdout [-p prmtop -c inpcrd]/[-pqr pqr]
```

Starting from the 2014 release, pbsa supports the free format pqr file. Once the pqr reading is enabled, the default Amber file reading and processing would be bypassed. Here is a brief description of the files mentioned above.

mdin *input* control data for the run.

mdout *output* user readable state info and diagnostics “-o stdout” will send output to stdout (to the terminal) instead of to a file.

prmtop *input* molecular topology, force field, atom and residue names, and (optionally) periodic box type.

inpcrd *input* initial coordinates and (optionally) velocities and periodic box size.

pqr *input* initial coordinates, atomic charges and radii in the free format pqr.

A few comments on the “free-formatted” pqr file used by *pbsa*. First all fields are delimited by spaces only. Second there is no strict format requirement as in a standard pdb file. This more liberal style is to accomodate pqr files of different origins. *pbsa* reads data on a per-line basis using the following format:

Tag AtomNumber AtomName ResidueName ChainID ResidueNumber XYZ Charge Radius

Tag A string specifying either ATOM or HETATM. Lines with other strings are ignored.

AtomNumber The sequence no of the atom, which is reset to start from 1.

AtomName The atom name.

ResidueName The residue name.

ChainID The chain ID of the atom, optional, which is ignored.

ResidueNumber The sequence no. of the residue, which is ignored.

XYZ The floating numbers representing the atomic coordiantes (in Angstrom).

Charge A float number providing the atomic charge (in electron).

Radius A float number providing the atomic radius (in Angstrom).

Finally it is worth to point out that it is apparently very hard to know whether the charge and radius fields are swapped as in the *Delphi* generated pqr file. Here we have assumed that the data are in the plain P.Q.R. order. Please make sure you are following the same convention in generating the pqr files.

6.2.2. Basic input options

The layout of the input file is in the same way as that of *Amber/sander* for backward compatibility with previous releases in *Amber*. The keywords are put in the the namelist of &cntrl for basic controls and &pb for more detailed manipulation of the numerical procedures. This subsection discusses the basic keywords, either retained from *sander* or newly created to invoke different energetic analyses. To reduce confusion most keywords from *sander* have been removed from the namelist so they can no longer be read since the current implementation in *pbsa* only performs single-structure calculations with the coordinates from **inpcrd** and exits. However, the current release is compatible with the **mdin** file generated with the *mmpbsa* script in previous releases in *Amber*. Users interested in energy minimization and molecular dynamics with the PB implementation are referred to *sander* in the release of *Amber*. Nevertheless, for purposes of validation and development, the atomic forces can be dumped out in a file when requested as described below.

The numerical electrostatic procedures can be turned on by setting IPB to either 1, 2 or 4. The flag IGB = 10 is phased out in this release. The numerical non-polar procedures can be turned on by setting INP to either 1 or 2. The backward compatible flag NPOPT is also phased out in this release.

imin Flag to run minimization. Both options give the same output energies though the output formats are slightly different. This option is retained from previous releases in the *Amber* package for backward compatibility. The current release of *pbsa* only supports single point energy calculation.

= 0 No minimization. Dynamics is available with *sander* and NAB.

= 1 Single point energy calculation. Default. Multiple-step PB minimization is also available with *sander* and NAB.

ntx Option to read the coordinates from the "inpcrd" file. Only options 1 and 2 are supported in this releases. Other options will cause *pbsa* to issue a warning though it does not affect the energy calculation.

= 1 X is read formatted with no initial velocity information. Default.

= 2 X is read unformatted with no initial velocity information.

ipb Option to set up a dielectric model for all numerical PB procedures. IPB = 1 corresponds to a classical geometric method, while a level-set based algebraic method is used when IPB $\geqslant 2$. The default IPB is 2.

= 0 No electrostatic solvation free energy is computed.

= 1 The dielectric interface between solvent and solute is built with a geometric approach.

= 2 The dielectric interface is implemented with the level set function. Use of a level set function simplifies the calculation of the intersection points of the molecular surface and grid edges and leads to more stable numerical calculations. Default.

= 4 The dielectric interface is also implemented with the level set function. However, the linear equations on the irregular points are constructed using the IIM. In this option, The dielectric constant do not need to be smoothed, that is, SMOOTHOPT is useless. Only the linear PB equation is supported, that is, NPBOPT = 0. And the different solvers are used to solve the generated linear equation set, that is, the meaning of SOLVOPT is changed as shown below.

inp Option to select different methods to compute non-polar solvation free energy.

= 0 No non-polar solvation free energy is computed.

= 1 The total non-polar solvation free energy is modeled as a single term linearly proportional to the solvent accessible surface area, as in the PARSE parameter set, that is, if INP = 1, USE_SAV must be equal to 0. See Introduction.

= 2 The total non-polar solvation free energy is modeled as two terms: the cavity term and the dispersion term. The dispersion term is computed with a surface-based integration method [173] closely related to the PCM solvent for quantum chemical programs.[175] Under this framework, the cavity term is still computed as a term linearly proportional to the molecular solvent-accessible-surface area (SASA) or the molecular volume enclosed by SASA. Default.

Once the above basic options are specified, *pbsa* can proceed with the default options to compute the solvation free energies with the input coordinates. Of course, this means that you only want to use default options for default applications.

More PB options described below can be defined in the &pb namelist, which is read immediately after the &cntrl namelist. We have tried hard to make the defaults for these parameters appropriate for calculations of solvated molecular systems. Please use caution when changing any default options.

6.2.3. Options to define the physical constants

epsin	Sets the dielectric constant of the solute region, default to 1.0. The solute region is defined to be the solvent excluded volume.
epsout	Sets the implicit solvent dielectric constant, default to 80. The solvent region is defined to be the space not occupied the solute region. i.e., only two dielectric regions are allowed in the current release.
epsmem	Sets the membrane dielectric constant. Only used if membraneopt > 0, does nothing otherwise. Value used should be between epsin and epsout or there may be errors. Previously spelled as epsmemb, which is being phased out. Defaults to 1.0.
smoothopt	Instructs PB how to set up dielectric values for finite-difference grid edges that are located across the solute/solvent dielectric boundary. **= 0** The dielectric constants of the boundary grid edges are always set to the equal-weight harmonic average of EPSIN and EPSOUT. **= 1** A weighted harmonic average of EPSIN and EPSOUT is used for boundary grid edges. The weights for EPSIN and EPSOUT are fractions of the boundary grid edges that are inside or outside the solute surface.[186] Default. **= 2** The dielectric constants of the boundary grid edges are set to either EPSIN or EPSOUT depending on whether the midpoints of the grid edges are inside or outside the solute surface.
istrng	Sets the ionic strength (in mM) for the PB equation. Default is 0 mM. Note the unit is different from that (in M) in the generalized Born methods implemented in *Amber*. Note also that we are only dealing with symmetrical solution, so the ionic strength should be equal to the square of the valence of the symmetrical ions times the ion concentration (in mM).
pbtemp	Temperature (in K) used for the PB equation, needed to compute the Boltzmann factor for salt effects; default is 300 K.
radiopt	Option to set up atomic radii. **= 0** Use radii from the prmtop file for both the PB calculation and for the NP calculation (see INP). **= 1** Use atom-type/charge-based radii by Tan and Luo [187] for the PB calculation. Note that the radii are optimized for *Amber* atom types as in standard residues from the *Amber* database. If a residue is built by *antechamber*, i.e., if GAFF atom types are used, radii from the prmtop file will be used. Please see [187] on how these radii are optimized. The procedure in [187] can also be used to optimize radii for nonstandard residues. These optimized radii can be read in if they are incorporated into the radii section of the prmtop file (of course via RADIOPT = 0). Default.
dprob	Solvent probe radius for molecular surface used to define the dielectric boundary between solute and solvent. DPROB = 1.4 by default.
iprob	Mobile ion probe radius for ion accessible surface used to define the Stern layer. Default to 2.0 Å.

sasopt Option to determine which kind of molecular surfaces to be used in the Poisson-Boltzmann implicit solvent model. Default is 0.

= 0 Use the solvent excluded surface as implemented by[185]

= 1 Use the solvent accessible surface. Apparently, this reduces to the van der Waals surface when the dprobe is set to zero.

= 2 Use the smooth surface defined by a revised density function.[188] This must be combined with IPB $\geqslant 2$.

saopt Option to compute the surface area of a molecule. Default is 0. Once the computation is enabled, the surface area will be reported in the output file with the subtitle “Total molecular surface”. Note that only the surface areas for the solvent excluded surface and the solvent accessible surface are supported in this release.

= 0 Do not compute any surface area.

= 1 Use the field-view method to compute the surface area.[164]

triopt Option to add trimer arc dots for a more accurate and lower memory mapping method of the analytical solvent excluded surface. See [185]

= 0 Trimer arc dots are not used.

= 1 Trimer arc dots are used. Default.

arcres *pbsa* uses a numerical method to compute solvent accessible arcs. See [185]. The ARCRES keyword gives the resolution (in the unit of Å) of dots used to represent these arcs, default to 0.25 Å. These dots are first checked against nearby atoms to see whether any of the dots are buried. The exposed dots represent the solvent accessible portion of the arcs and are used to define the dielectric constants on the grid edges. It should be pointed out that ARCRES should be reduced to (0.125 Å) when the TRIOPT option is turned off to achieve a similar accuracy in the reaction field energies. More generally, ARCRES should be set to $max(0.125\ \text{Å}, 0.5h)$ when the TRIOPT option is turned on, or $max(0.0625\ \text{Å}, 0.25h)$ when the TRIOPT option is turned off (h is the grid spacing).

6.2.4. Options for Implicit Membranes

membraneopt Option to turn implicit membrane on and off. Membrane is implemented as a slab like region with same dielectric constant as solute. Other membrane setup schemes will be made available in the future.

= 0 No implicit membrane used (default).

= 1 Use a slab-like implicit membrane.

mprob Membrane probe radius in Å, default to 2.70. This is used to specify the highly different lipid molecule accessibility versus that of the water.

mthick Membrane thickness in Å, default to 40.0. This is different from the previous default of 20 Å.

mctrdz Membrane center in Å in the z direction. Default is 0 - membrane centered at the center of the protein.

poretype Turn on and off the automatic depth-first search method to identify the pore.

= 0 Do not turn on the pore searching algorithm.

= 1 Turn on the pore searching algoirthm.

poreradius Controls the radius, in Å, of the cylindrical exclusion region. This is no longer needed given the automatic pore searching algorithm.

6.2.5. Options to select numerical procedures

npbopt	Option to select the linear or the full nonlinear PB equation. **= 0** Linear PB equation is solved. Default. **= 1** Nonlinear PB equation is solved.
solvopt	Option to select iterative solvers. **= 1** Modified ICCG or Periodic (PICCG) if bcopt = 10 is. Default. If IPB = 4, an algebraic multigrid solver is used. **= 2** Geometric multigrid. A four-level v-cycle implementation is applied. Each dimension of the finite-difference grid is 2^4×n-1. If IPB = 4, preconditioned GMRES. **= 3** Conjugate gradient (Periodic version available under bcopt = 10). This option requires a large MAXITN to converge. If IPB = 4, preconditioned BiCG. **= 4** SOR. This option requires a large MAXITN to converge. **= 5** Adaptive SOR. This is only compatible with NPBOPT = 1. This option requires a large MAXITN converge. **= 6** Damped SOR. This is only compatible with NPBOPT = 1. This option requires a large MAXITN to converge.
accept	Sets the iteration convergence criterion (relative to the initial residue). Default to 0.001.
maxitn	Sets the maximum number of iterations for the finite difference solvers, default to 100. Note that MAXITN has to be set to a much larger value, like 10,000, for the less efficient solvers, such as conjugate gradient and SOR, to converge.
fillratio	The ratio between the longest dimension of the rectangular finite-difference grid and that of the solute. Default is 2.0. It is suggested that a larger FILLRATIO, for example 4.0, be used for a small solute, such as a ligand molecule. Otherwise, part of the small solute may lie outside of the finite-difference grid, causing the finite-difference solvers to fail.
space	Sets the grid spacing for the finite difference solver; default is 0.5 Å.
nbuffer	Sets how far away (in grid units) the boundary of the finite difference grid is away from the solute surface; default is 0 grids, i.e., automatically set to be at least a solvent probe or ion probe (diameter) away from the solute surface.
nfocus	Set how many successive FD calculations will be used to perform an electrostatic focussing calculation on a molecule. Default to 2, the maximum. When NFOCUS = 1, no focusing is used. It is recommended that NFOCUS = 1 when the multigrid solver is used.
fscale	Set the ratio between the coarse and fine grid spacings in an electrostatic focussing calculation. Default to 8.
npbgrid	Sets how often the finite-difference grid is regenerated; default is 1 step. For molecular dynamics simulations, it is recommended to be set to at least 100. Note that the PB solver effectively takes advantage of the fact that the electrostatic potential distribution varies very slowly during dynamics simulations. This requires that the finite-difference grid be fixed in space for the code to be efficient. However, molecules do move freely in simulations. Thus, it is necessary to set up the finite-difference grid once in a while to make sure a molecule is well within the grid.

6.2.6. Options to compute energy and forces

ENEOPT is the option to set a method to compute electrostatic energy and forces, and DBFOPT is phased out in this release.

bcopt Boundary condition options.

= 1 Boundary grid potentials are set as zero, i.e. conductor. Total electrostatic potentials and energy are computed.

= 5 Computation of boundary grid potentials using all grid charges. Total electrostatic potentials and energy are computed. Default.

= 6 Computation of boundary grid potentials using all grid charges. Reaction field potentials and energy are computed with the charge singularity free formulism.[182]

= 10 Periodic boundary condition is used. Total electrostatic potentials and energy are computed. Can be used with SOLVOPT = 1, 2, 3, or 4 and IPB = 1 or 2. It should only be used on charge-neutral systems. If the system net charge is detected to be nonzero, it will be neutralized by applying a small neutralizing charge on each grid (i.e. a uniform plasma) before solving.

eneopt Option to compute total electrostatic energy and forces.

= 1 Compute total electrostatic energy and forces with the particle-particle particle-mesh (P3M) procedure outlined in Lu and Luo.[169] In doing so, energy term EPB in the output file is set to zero, while EEL includes both the reaction field energy and the Coulombic energy. The van der Waals energy is computed along with the particle-particle portion of the Coulombic energy. The electrostatic forces and dielectric boundary forces can also be computed.[169] This option requires a nonzero CUTNB and BCOPT = 5.

= 2 Use dielectric boundary surface charges to compute the reaction field energy. Default. Both the Coulombic energy and the van der Waals energy are computed via summation of pairwise atomic interactions. Energy term EPB in the output file is the reaction field energy. EEL is the Coulombic energy.

= 3 Similar to the first option above, a P3M procedure is applied for both solvation and Coulombic energy and forces for larger systems.

= 4 Similar to the third option above, a P3M procedure for the full nonlinear PB equation is applied for both solvation and Coulombic energy and forces for larger systems. A more robust and clean set of routines were used for the P3M and dielectric surface force calculations.

frcopt Option to compute and output electrostatic forces to a file named *force.dat* in the working directory.

= 0 Do not compute or output atomic and total electrostatic forces. This is default.

= 1 Reaction field forces are computed by trilinear interpolation. Dielectric boundary forces are computed using the electric field on dielectric boundary. The forces are output in the unit of kcal/-mol·Å.

= 2 Use dielectric boundary surface polarized charges to compute the reaction field forces and dielectric boundary forces [183] The forces are output in the unit of kcal/mol·Å.

= 3 Reaction field forces are computed using dielectric boundary polarized charge. Dielectric boundrary forces are computed using the electric field on dielectric boundary. [189] The forces are output in the unit of kcal/mol·Å.

scalec Option to compute reaction field energy and forces.

= 0 Do not scale dielectric boundary surface charges before computing reaction field energy and forces. Default.

= 1 Scale dielectric boundary surface charges using Gauss's law before computing reaction field energy and forces.

cutfd	Atom-based cutoff distance to remove short-range finite-difference interactions, and to add pairwise charge-based interactions, default is 5 Å. This is used for both energy and force calculations. See Eqn (20) in Lu and Luo.[169]
cutnb	Atom-based cutoff distance for van der Waals interactions, and pairwise Coulombic interactions when ENEOPT = 2. Default to 0. When CUTNB is set to the default value of 0, no cutoff will be used for van der Waals and Coulombic interactions, i.e., all pairwise interactions will be included. When ENEOPT = 1, this is the cutoff distance used for van der Waals interactions only. The particle-particle portion of the Coulombic interactions is computed with the cutoff of CUTFD.
nsnba	Sets how often atom-based pairlist is generated; default is 1 step. For molecular dynamics simulations, a value of 5 is recommended.

6.2.7. Options for visualization and output

phiout *pbsa* can be used to output spatial distribution of electrostatic potential for visualization.

= 0 No potential file is printed out. Default.

= 1 Electrostatic potential is printed out in a file named pbsa.phi in the working directory. Please refer to examples in the next section on how to display electrostatic potential on molecular surface.

phiform Controls the format of the electrostatic potential file.

= 0 The electrostatic potential (kT/mol·e) is printed in the *Delphi* binary format. Default.

= 1 The electrostatic potential (kcal/mol·e) is printed in the *Amber* ASCII format.

= 2 The electrostatic potential (kcal/mol·e) is printed in the DX volumetric data format for use with *VMD*.

outlvlset *pbsa* can be set to write the total level set, used in locating interfaces between regions of differing dielectric constant, to a DX format volumetric data file. This option will control printing of the total level set (i.e. both solute-solvent and membrane level sets combined if membrane present)

= false No level set file printed out. Default.

= true Level set printed out in a file named pbsa_lvlset.dx

outmlvlset *pbsa* can be set to write the membrane level set, used in locating interfaces between regions of differing dielectric constant, to a DX format volumetric data file. This option controls printing a separate file for the membrane level set. Does nothing if membraneopt is not turned on.

= false No level set file printed out. Default.

= true Level set printed out in a file named pbsa_lvlset.dx

npbverb When set to 1, turns on verbose mode in *pbsa*; default is 0.

6.2.8. Options to select a non-polar solvation treatment

decompopt Option to select different decomposition schemes when INP = 2. See [173] for a detailed discussion of the different schemes. The default is 2, the σ decomposition scheme, which is the best of the three schemes studied.[173] As discussed in Ref. [173], DECOMPOPT = 1 is not a very accurate approach even if it is more straightforward to understand the decomposition.

= 1 The 6/12 decomposition scheme.

= 2 The σ decomposition scheme. Default

= 3 The WCA decomposition scheme.

use_rmin The option to set up van der Waals radii. The default is to use *rmin* to improve the agreement with TIP3P [173].

= 0 Use atomic van der Waals σ values.

= 1 Use atomic van der Waals *rmin* values. Default.

sprob Solvent probe radius for solvent accessible surface area (SASA) used to compute the dispersion term, default to 0.557 Å in the σ decomposition scheme as optimized in Ref. [173] with respect to the TIP3P solvent and the PME treatment. Recommended values for other decomposition schemes can be found in Table 4 of [173]. If USE_SAV = 0 (see below), SPROB can be used to compute SASA for the cavity term as well. Unfortunately, the recommended value is different from that used in the dispersion term calculation as documented in Ref. [173] Thus two separate *pbsa* calculations are needed when USE_SAV = 0, one for the dispersion term and one for the cavity term. Therefore, please carefully read Ref. [173] before proceeding with the option of USE_SAV = 0. Note that SPROB was used for ALL three terms of solvation free energies, i.e., electrostatic, attractive, and repulsive terms in previous releases in *Amber*. However, it was found in the more recent study [173] that it was impossible to use the same probe radii for all three terms after each term was calibrated and validated with respect to the TIP3P solvent. [173, 187]

vprob Solvent probe radius for molecular volume (the volume enclosed by SASA) used to compute non-polar cavity solvation free energy, default to 1.300 Å, the value optimized in Ref. [173] with respect to the TIP3P solvent. Recommended values for other decomposition schemes can be found in Tables 1-3 of Ref. [173].

rhow_effect Effective water density used in the non-polar dispersion term calculation, default to 1.129 for DECOMPOPT = 2, the σ scheme. This was optimized in Ref. [173] with respect to the TIP3P solvent in PME. Optimized values for other decomposition schemes can be found in Table 4 of Ref. [173].

use_sav The option to use molecular volume (the volume enclosed by SASA) or to use molecular surface (SASA) for cavity term calculation. The default is to use the molecular volume enclosed by SASA. Recent study shows that the molecular volume approach transfers better from small training molecules to biomacromolecules.

= 0 Use SASA to estimate cavity free energy.

= 1 Use the molecular volume enclosed by SASA. Default.

cavity_surften The regression coefficient for the linear relation between the total non-polar solvation free energy (INP = 1) or the cavity free energy (INP = 2) and SASA/volume enclosed by SASA. The default value is for INP = 2 and set to the best of three tested schemes as reported in Ref. [173], i.e. DECOMPOPT = 2, USE_RMIN = 1, and USE_SAV = 1. See recommended values in Tables 1-3 for other schemes.

cavity_offset The regression offset for the linear relation between the total non-polar solvation free energy (INP = 1) or the cavity free energy (INP = 2) and SASA/volume enclosed by SASA. The default value is for INP = 2 and set to the best of three tested schemes as reported in Ref. [173], i.e. DECOMPOPT = 2, USE_RMIN = 1, and USE_SAV = 1. See recommended values in Tables 1-3 for other schemes.

maxsph *pbsa* uses a numerical method to compute solvent accessible surface area.[173] MAXSPH variable gives the approximate number of dots to represent the maximum atomic solvent accessible surface, default to 400. These dots are first checked against covalently bonded atoms to see whether any of the dots are buried. The exposed dots from the first step are then checked against a non-bonded pair list with a cutoff distance of 9 Å to see whether any of the exposed dots from the first step are buried. The exposed dots of each atom after the second step then represent the solvent accessible portion of the atom and are used to compute the SASA of the atom. The molecular SASA is simply a summation of the atomic SASA's. A molecular SASA is used for both PB dielectric map assignment and for NP calculations.

6.2.9. Options to enable active site focusing

Active site focusing is an extension to the electrostatic focusing method. Electrostatic focusing can be regarded as a multi-level FDPB calculation (two levels currently implemented) in which a coarse-grid solution is conducted to set up the boundary condition for the requested fine-grid solution. In the original implementation of electrostatic focusing, the fine grid always covers all the solute atoms. However in the enhanced implementation, the fine grid is allowed to cover only a local region of interest, such as an enzyme active site or ligand docking site. In such applications, most or all of the protein atoms are held frozen during a calculation while only the active site side chain and the substrate ligand are allowed to move. In principle, energies computed with the local electrostatic focusing method should correlate with those computed with the original electrostatic focusing method if the movable substrate/ligand atoms are well within the local region of interest. The “active site” or the local region is specified as a rectangular box by the following six variables:

xmax	The upper boundary of the box in x direction.
xmin	The lower boundary of the box in x direction, XMAX has to be greater than XMIN.
ymax	The upper boundary of the box in y direction.
ymin	The lower boundary of the box in y direction, YMAX has to be greater than YMIN.
zmax	The upper boundary of the box in z direction.
zmin	The lower boundary of the box in z direction, ZMAX has to be greater than ZMIN.

Of course, these keywords are zero by default, i.e. the original electrostatic focusing would be invoked if these keywords remain to be the default value of zero.

6.2.10. Options to enable multiblock focusing

In order to handle large molecular systems with typical computer hardwares available to our end users, the basic principle of the electrostatic focusing discussed in the previous subsection is extended for the multiblock electrostatic focusing method. Briefly, the time-limiting step of FDPB, the fine-grid calculation, is divided into a series of smaller jobs, with each solving only a small local region of a large molecular system. Once all the smaller jobs are finished, the solutions are combined to obtain the final energy for the large molecular system. Note that this is an approximated method, just like the original electrostatic focusing method. In this implementation, overlapping/padding grid points are used to preserve accuracy. Most of the settings for this feature are hidden from end users except the dimensions of the multi-blocks. [190]

Before your production runs, please activate NPBVERB = 1 and check in the mdout file to see if your multiblock settings are indeed reasonable. Here are some hints. First, the blocksize should be around 64^3 to 96^3 for typical computers with 8GB memory. Secondly, the grid dimension, *xm*, should be divisible by $(ngrdblkx-1)$, or slightly larger, for the x direction. The same applies for y and z directiosn as well. Keep in mind that the incentive for choosing this method is to be able to work with large systems on typical computer hardwares.

ngrdblkx	The number of fine-grid points for a focusing block in x direction, $(ngrdblkx-1)$ should be divisible by FSCALE.
ngrdblky	The number of fine-grid points for a focusing block in y direction, $(ngrdblky-1)$ should be divisible by FSCALE.
ngrdblkz	The number of fine-grid points for a focusing block in z direction, $(ngrdblkz-1)$ should be divisible by FSCALE.

pbsa can also be run in parallel environment with *pbsa.MPI* executable but for multiblock focusing only. Do make sure that the number of nodes is less than the number of focusing blocks.

6.3. Example inputs and demonstrations of functionalities

6.3.1. Single-point calculation of solvation free energies

Normally the default *pbsa* options are capable of dealing with most situations. Users should be fully aware of the meaning of an option before they change its default value. In all the following example inputs, only the options that are different from their default values will be shown, and the explanations on the changes will be given in detail. Here is a sample input file that might be used to perform single structure calculations.

```
Sample single point PB calculation
&cntrl
/
&pb
npbverb=1, istrng=150, fillratio=1.5, saopt=1,
/
```

Note that NPBVERB = 1 above. This generates much detailed information in the output file for the PB and NP calculations. A useful printout is atomic SASA data for both PB and NP calculations which may or may not use the same atomic radius definition. Since the FD solver for PB is called twice to perform electrostatic focus calculations, two PB printouts are shown for each single point calculation. For the PB calculation, a common error message can be generated when FILLRATIO is set to the default value of 2.0 for small molecules. This may cause a solute to lie outside of the focusing finite-difference grid.

In this example INP is not set and equal to the default value of 2, which calls for non-polar solvation calculation with the new method that separates cavity and dispersion interactions. The EDISPER term gives the dispersion solvation free energy, and the ECAVITY term gives the cavity solvation free energy. The default options for the NP calculation are set to the recommended values for the σ decomposition scheme and to use molecular volume to correlate with cavity free energy. You can find recommended values for other decomposition schemes and other options in Tables 1-4 of Ref. [173]. If INP is set to 1, the ECAVITY term would give the total non-polar solvation free energy.

Finally, a few words on the RADIOPT option, set to the default value of 1 instructing PB to use the optimized values instead of reading the radii from the prmtop file. Starting this release, the RADIOPT option only controls the radius definition for the PB calculation. The INP=2 calculation automatically uses the default values, such as atomic radii and solvent probes as optimized in Ref. [173]. On the other hand, the INP=1 calculation is allowed to use whatever radii that a user decides to use.

The ion strength option ISTRNG is set to 150 in unit mM, a typical value for a physiological environment. The FILLRATIO option is set to 1.5 because the biomolecule is relatively large. We set saopt to 1 because we need the information of the molecular surface area (the molecular surface is defined as the solvent excluded surface since SASOPT is set to its default value 0).

6.3.2. Implicit membrane model

pbsa now supports inclusion of an implicit membrane region in implicit solvation calculations. This feature is enabled by setting MEMBRANEOPT to 1 (default value is 0, for off). The membrane will extend the solute dielectric region to include a slab-like planar region running parallel to the xy plane. The thickness is controlled by the MTHICK option. The default is 40 Å. The membrane region will be centered on the protein center by default, and can be set to a user-provided value using the MCTRDZ option (default is 0). Neither option will have any effect unless MEMBRANEOPT is set to 1. The dielectric constant can be controlled using epsmem. We set the membrane interior dielectric constant to a value of 4.0 in this example. This is four times that of the solute which defaults to 1 (same as vacuum). The value of epsmem should always be set to a value greater than or equal to EPSIN (solute dielectric constant) and less than EPSOUT (solvent dielectric constant). These default to 1.0 and 80.0 respectively.

When using the implicit membrane model, we recommend SASOPT=0, i.e. the classical solvent excluded surface, due to its better numerical behavior. When running with the default options, the program will compute solvent excluded surfaces both with the water probe (DPROB=1.40 by default) and the membrane probe

(MPROB=2.70 by default). This setting was found to be consistent with the explicit solvent MD simulations. It is also suggested that periodic boundary conditions be used to avoid unphysical edge effects. This is supported in all linear solvers. In the following we choose Periodic Incomplete Cholesky Conjugate Gradient (PICCG). So we set IPB = 1, BCOPT = 10, and SOLVOPT = 1 (default). In addition, ENEOPT needs to be set to 1 because the charge-view method (ENEOPT = 2) is not supported for this application.

```
Sample single point PB calculation with membrane region
&cntrl
ipb=1, inp=0
/
&pb
radiopt=0, nfocus=1, maxitn=200,
bcopt=10, eneopt=1, solvopt=1,
sasopt=0, membraneopt=1, epsmem = 4.0
outlvlset=true, outmlvlset=true
/
```

The MAXITN option is set to a bigger value, 200, than the default one, 100, because the linear solvers, when applied to periodic boundary conditions, seem to require slightly more iterations than non-periodic solvers to converge.

To aid in visualization of the dielectric model, the level set function, which is used to locate the interfacial surfaces between regions of differing dielectric constant, can be written to output files. Output of the total level set function, including both the solute-solvent and membrane contributions, can be written to a DX formatted volumetric data file by setting the OUTLVLSET option to "true". The membrane contribution can be written to a separate file by setting the OUTMLVLSET option to "true". This may take a good deal of extra time, so be sure to leave it off if you don't want / need to visualize the levelset surface. Accordingly, NFOCUS is set to 1 because of the use of periodic boundary condition.

Finally, if calculations need to be performed on a protein with a solvent-filled channel region, this region will be identified automatically by setting PORETYPE=1.

6.3.3. Single point calculation of forces

Since *pbsa* is released for single point calculations in *AmberTools*, no energy minimization or molecular dynamics is supported. However, the PB procedure can be invoked to print out the numerical electrostatic forces for developmental purposes. Here is a sample input:

```
Sample PB force computation
&cntrl
inp=0
/
&pb
npbverb=1, radiopt=0, frcopt=2
/
```

Note that INP is set to 0 to turn off non-polar solvation interactions. RADIOPT = 0 means the atomic radii from the topology files will be used. FRCOPT is set to 2, *i.e.,* induced surface charges are used to compute the electrostatic energy and forces. Since CUTNB is equal to the default value of zero, an infinite cutoff distance is used for both Coulombic and van der Waals interactions.

6.3.4. Comparing with *Delphi* results

Under identical condition, *pbsa* is highly consistent with *Delphi* in term of computed reaction field energies. In this subsection, we briefly go over the details on how you can obtain comparable energies from both programs. Apparently, you need coordinates, atomic charges, and atomic radii that have exactly the same numerical values but in both the *Amber* format and the *Delphi* format, i.e., the pqr format.

For a *Delphi* computation with the following input parameters:

```
salt=0.150
ionrad=2.0
exdi=80.0
indi=1.0
scale=2.0
prbrad=1.5
perfil=50
bndcon=4
linit=1000
```

A comparable computation in *pbsa* can be obtained by using the following input file:

```
Sample PB for delphi comparison
&cntrl
ipb=1, inp=0
/
&pb
istrng=150, ivalence=1, iprob=2.0, dprob=1.5,
radiopt=0, bcopt=5, smoothopt=2, nfocus=1,
/
```

IPB is set to 1 to make sure *pbsa* is using the exactly same surface definition as *Delphi*. Note that the values of *exdi*, *indi*, *prbrad*, and *ionrad* in *Delphi* should be consistent with the values of EPSOUT, EPSIN, DPROB, and IPROB in *pbsa*, respectively. In *Delphi salt=0.150* is set in the unit of M, while in *pbsa* ISTRNG = 150 is in the unit of mM. In *Delphi* the grid spacing is set as the number of grids per Å, i.e., *scale=2.0*, while in *pbsa* the grid spacing is set straight in Å as SPACE = 0.5. In *Delphi* the grid dimension is set as percentage of the solute dimension over the grid dimension, i.e., *perfil=50*, which is equivalent to the ratio of solute dimension over grid dimension set as FILLRATIO = 2 in *pbsa*. Finally, *Delphi* sets the boundary condition by *bndcon=4* and *pbsa* sets the boundary condition as BCOPT = 5; both programs mean to use the Debye-Huckel limitation behavior for each atomic charged sphere. There are additional options in *pbsa* that do not have corresponding counterparts in Delphi. For example, SMOOTHOPT is used to instruct the program to use a specific dielectric boundary smoothing option, which is equivalent to that used in *Delphi* when set to 2. (see Section 5.2.3).

6.4. Visualization functions in *pbsa*

pbsa can produce volumetric data files to allow visualization of electrostatic potential and level set maps. There are two points to note before continuing.

1. The data files generated can become quite large if small grid spacings are used since they will scale as the cube of the inverse of grid spacing
2. Unless singularity removal methods are used, the potential at grid nodes corresponding to atom centers may be quite large when compared to the potential at the molecular / atomic surface. This will often result in poor contrast during visualization of the potential map, particularly when it is used as a color map for a molecular surface.

These two points should be kept in mind when determining grid spacing. For visualization purposes, a grid spacing of about one angstrom should provide good results. If finer spacing is needed, singularity removal (BCOPT = 6) can be used to prevent poor contrast that could result from the presence of singularities. Lastly, when using grid spacings of 0.5 Å or lower, the output files may become quite large (tens, or even hundreds of megabytes each) and may take a significant amount of time (up to several seconds each) to generate.

6.4.1. Visualization of electrostatic potential using *PyMol*

pbsa can produce an electrostatic potential map for visualization in *PyMol* when setting PHIOUT = 1. By default, *pbsa* outputs a file pbsa.phi in the *Delphi* binary format. The sample input file is listed below:

```
Sample PB visualization input
&cntrl
inp=0
/
&pb
npbverb=1, space=1.,
phiout=1, phiform=0
/
```

To be consistent with the surface routine of *PyMol*, the option PHIOUT = 1 instructs *pbsa* to use the radii as defined in *PyMol*. The finite-difference grid is also set to be cubic as in *Delphi*. The default DPROB value is equal to that used in *PyMol*, 1.4 Å. A large grid spacing, e.g. 1 Å or higher, is recommended for visualization purposes, as commented above.

Here is an example of loading the potential map in *PyMol.* First load the molecule in the form of prmtop and inpcrd. In our case we need to rename our prmtop file to molecule.top and inpcrd file to molecule.rst and load the molecule with commands

```
PyMol> load molecule.top
PyMol> load molecule.rst
```

The molecule will appear as an object “molecule”. Next display the surface of the molecule in the *PyMol* menu by clicking “S” and then select surface. Now import the potential map generated by *pbsa* with the command in *PyMol*

```
PyMol> load pbsa.phi
```

to create a value map object called “pbsa”. After this, create a value ramp called e_lvl from the potential map with the command

```
PyMol> ramp_new e_lvl, pbsa, [-7, 0, 7]
```

You can assign surface_color to the e_lvl ramp with the command

```
PyMol> set surface_color, e_lvl, molecule
```

This will display the surface with the color scale according to the potential. You can adjust the value scale, such as [-5, 0, 5], to change the color scale and use “rebuild” command to redraw the surface.

6.4.2. Writing electrostatic potential to DX format volumetric data file

To visualize the *pbsa* potential using *VMD*, you will need to set the output to DX format by changing PHIFORM = 0 to PHIFORM = 2.

```
Sample PB visualization input
&cntrl
inp=0
/
&pb
npbverb=1, space=1., sasopt=2,
phiout=1, phiform=2
/
```

The program will now generate a file called pbsa_phi.dx. This format should be automatically recognized by VMD. It can be either loaded directly into your molecule or as a separate file.

6.4.3. Loading DX format electrostatic potential data in *VMD*

1. go to the “File” menu in the *VMD* Main window.
2. Select “New Molecule...”.
    - This will bring up the “Molecule File Browser” window
3. Click on the “Browse...” button in the “Molecule File Browser” window
4. Select the file “pbsa_phi.dx” that was generated by *pbsa* using the file selection dialogue that pops up.
    - The “Determine file type:” drop down menu should now read “DX”.
5. Click the “Load” button.

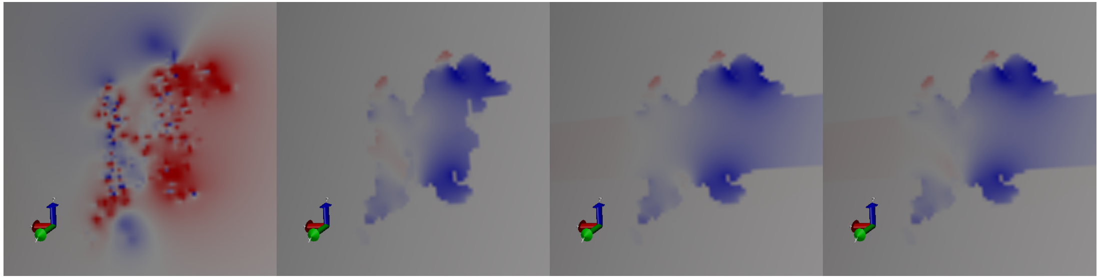
VMD will, by default, display the data with an isosurface representation.

6.4.4. Changing the representation model

1. Select “Representations...” from the “Graphics” menu in the “*VMD* Main” window
    - The “Graphical Representations” window should pop up
2. Select the object corresponding to the volumetric data you loaded from the “Selected Molecule” pull down menu
3. Click on the representation you wish to change
    - There should be one present for the isosurface being displayed
4. Click on the “Draw style” tab if it is not already selected
5. Select “Volume” from the “Coloring Method” pull down menu if it is not already chosen
    - Another pull down menu will appear next to it.
    - If you have multiple data files loaded for the same object you can choose which is used to color your chosen draw method representation here
6. The “Drawing Method” pull down menu will let you choose a different visual representation model.
    - To directly visualize potential data, use either “Isosurface” or “Volume Slice”
    - *VMD* can also be used to visualize the corresponding electric field by choosing “Field Lines”.

Displayed below are Volume Slice representations of electrostatic potential maps generated for an aquaporin system. Computations were run using the periodic conjugate gradient solver for a 1 Å grid spacing, and FILLRATIO of 2.0. For the systems using implicit water, the charge singularity removal methodology was used.

From Left to right: Vacuum, Water only, Water and 20 Å slab-like membrane, Water and 20 Å slab-like membrane with 6 Å cylindrical channel region removed.



Often, the data ranges will not be consistent between potential distributions for different implicit solvent setups. E.g. the range of the electrostatic values seen for vacuum will likely be larger than the range for implicit water. The range of values displayed can be set manually to provide consistent color scaling for comparison.

6.4.5. Adjusting the color scale of the color map

1. Select “Colors...” from the “Graphics” menu in the “*VMD* Main” window
    - This should cause the “Color Controls” window to pop up
2. Select the “Color Scale” tab
    - The color scheme can be selected from the “Method” pull down menu
    - The “Offset” and “Midpoint” sliders can be used to adjust the scaling of the color map.
        - If singularities are present, it may be difficult to get a good scaling for volume maps generated with fine grid spacings. In this case, either re-run with singularity removal on, or set the color scale range manually as shown in the next section.

When singularity removal is not employed, the presence of singularities will cause the range of the electrostatic potential distribution near the atom centers to be much wider than near the molecular surface. This typically results in very poor contrast particularly for implicit solvent since the high dielectric constant in the solvent region will amplify the effect. This can be compensated for by manually setting the Color Scale Data Range.

6.4.6. Changing the color scale range

1. Select desired representation to modify
2. Select “Volume” Coloring Method and Select the desired volumetric map to rescale from the pull down menu.
    - Each time you change the volumetric map being displayed, you will need to repeat this, so it is a good idea to make multiple representations for each potential data set rather than switching between them on the same representation.
3. Select the “Trajectory” tab
4. You should see the automatically computed range in the “Color Scale Data Range:” boxes. The left hand box controls the minimum value for the range, the right hand box controls the maximum value for the range.
5. Set the minimum and maximum values as needed to improve the contrast. Often the inner 10% to 30% of the total (automatic) range will give good contrast for a one angstrom grid spacing.
6. Click on the “Set” button when you are finished
7. To return to the automatic scaling that was originally calculated by *VMD*, click the “Autoscale” button.

Electrostatic potential data can also be used as a color map for other drawing methods. You will need to first load the data into the molecule you wish to display.

6.4.7. Loading electrostatic potential data into an existing molecule

The names of the files are used as labels, so it is useful to rename them from “pbsa_phi.dx” to something more descriptive before loading.

1. Select the molecule you wish to display the potential color map on in the “*VMD* Main” window
2. Go to the “File” menu in the *VMD* Main window.
3. Select “Load Data Into Molecule...”.
    - This will bring up the “Molecule File Browser” window
4. Click on the “Browse...” button in the “Molecule File Browser” window

5. Select the file “pbsa_phi.dx” that was generated by pbsa using the file selection dialogue that pops up.
   - The “Determine file type:” drop down menu should now read “DX”.
6. Click the “Load” button.

The data should now be loaded into the molecule you selected.

6.4.8. Using the electrostatic potential data as a color map

Once you have loaded a volumetric data file into a molecule, it can be used to generate a color map for any representations of that molecules model.

1. Open the “Graphical Representations” window if it is not already open
   - Select “Representations...” from the “Graphics” menu in the “*VMD* Main” window
2. Select the molecule you loaded the data into from the “Selected Molecule” pull down menu
3. Select the representation you wish to map the potential color map onto
4. Select the “Draw Style” tab if it is not already selected
5. Select “Volume” from the “Coloring Method” pull down menu
   - Another pull down menu should appear next to it
   - Choose the selection that corresponds to the data you just loaded, it should be the last one on the list if it is the last one that was loaded.

VMD will attempt to automatically scale the color mapping used for Volumetric data that you load. The color scale may be manually adjusted if needed (see previous section)

6.4.9. Loading and displaying the level set map

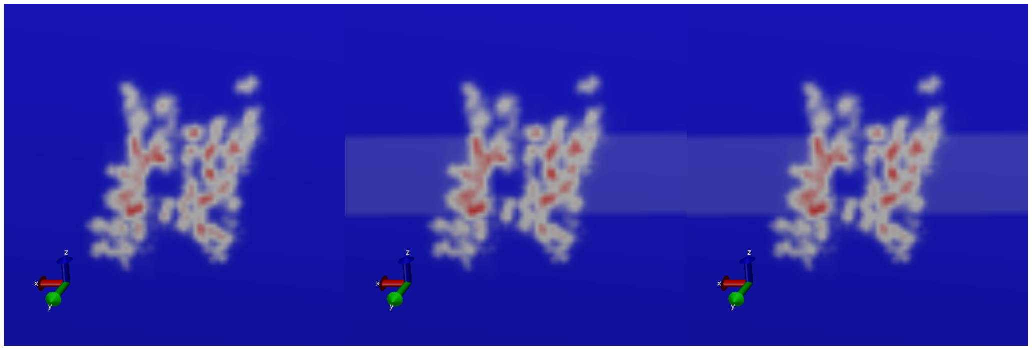
The level set used by *pbsa* to model the solute - solvent interface can be written to an output file in DX format by setting OUTLVLSET to “true” in the input file.

```
Sample PB visualization input
&cntrl
inp=0
/
&pb
npbverb=1, space=1., sasopt=2,
phiout=1, phiform=2,
outlvlset=true
/
```

The level set will be written to a DX format volumetric data file named “pbsa_lvlset.dx”. This file can be used to visualize the corresponding molecular surface. The level set file is loaded into *VMD* in the same manner as an electrostatic potential data file. Cross sections can be viewed using the “Volume Slice” representation.

Shown below are the level sets for the aquaporin systems shown previously (no level set is shown for vacuum as there is no dielectric interface being modeled in that system)

From left to right: Water, Water + Slab-like membrane, Water + Membrane with pore region



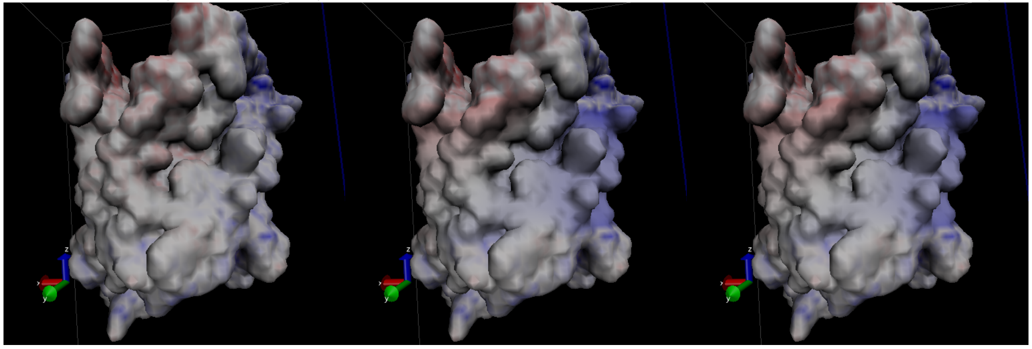
6.4.10. Visualizing the molecular surface as an isosurface of the level set

The level set is constructed such that the molecular surface is the locus of all points where the level set is zero. This allows us to use the Isosurface representation in *VMD* to display the solvent excluded surface by setting the "Isovalue" to 0. Alternatively, if we wish to view the potential just outside the surface, we can set the "Isovalue" to a number slightly higher than 0. E.g. 0.1 or 0.01.

1. Load the level set data file into the molecule.
   - This is done using the same procedure as loading an electrostatic potential data file, but the level set data file will be chosen instead of the potential data file.
2. Create a new Isosurface representation in the "Graphical Representations" window.
3. Select the volume map for the level set from the pull down menu
4. Choose an "Isovalue" at or slightly above 0.
5. Using the "Coloring Method" pull down menu, you may also use a previously loaded electrostatic potential data file as a color map by selecting "Volume" and then selecting the appropriate volume map from the pull down menu that appears.
   - *VMD* will automatically assign color scale range every time.
   - To compare multiple potential maps, it is often desirable to use the same color scale range for each. The best way to do this is to make a new representation for each potential map and manually assign the same color scale range to be identical for each (see previous section)

The examples below were generated for Aquaporin (1IH5 in the protein data bank) under various implicit solvent options using a FILLRATIO of 2.0, grid spacing of 1Å. For each calculation, the periodic conjugate gradient solver with singularity removal was used. The level set for the system modeling implicit water was used to build the isosurfaces. The electrostatic potential data files were then overlayed as color maps with the color scale ranges set to [-80000,80000].

From Left to right: Water only, Water + Slab Like Membrane, Water + Membrane with 6Å cylindrical pore.



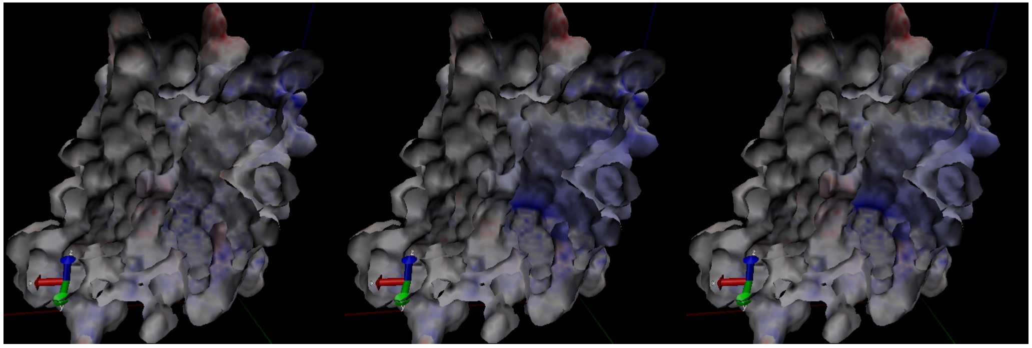
6.4.11. Visualizing interior channels, voids, and solvent pockets

One of the common roles for membrane proteins is to act as a transmembrane channel, to allow specific substance to pass from one side of a membrane to another. Features such as solvent / ion channels or internal voids will often be occluded from view by the exterior surface. One option that can allow these to be viewed is to use the clipping plane tool in VMD.

1. Open the "Exensions" pull down menu in the "*VMD* Main" window and go to the "Visualization" submenu and select "Clipping Plane Tool".
2. The "Clip Tool" window should pop up.
3. The "Distance" slider allows clipping to be set
4. The "Normal" slider sets the normal of the clipping plane.
    - The "flip" button on the right will let you clip from front to back, which will be useful to clip the occluding exteriro surface from the view and reveal the interior.

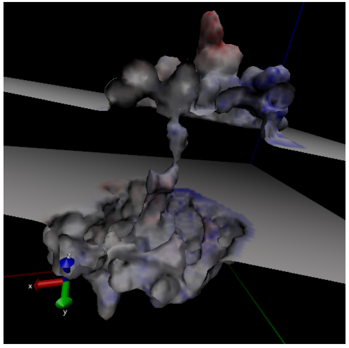
The clipping tool was used to reveal the internal pore region for the aquaporin system setups used in the previous section.

From Left to right: Water only, Water + Slab like Membrane, Water + Membrane with pore region excluded.



As an alternative, the level set map generated using PORTYPE=1 with the implicit membrane option will allow a cylindrical region to be excluded from the membrane level set. The corresponding isosurface will show any interior cavities or voids which fall within this region for isovalues at or slightly above 0 (since the level set at the membrane-solute interface will be below 0). See the previous section for details on writing and loading the level set file.

Shown below is the level set isosurface for the aquaporin system with implicit water plus a membrane with a cylindrical region removed. The corresponding potential data was again overlayed as a color map. The surface of the channel region, and the membrane-solvent interface planes are now clearly visible.



6.4.12. Importing / Modifying Atomic Radii to VMD from the prmtop file

Currently, VMD does not support loading radii for atoms directly from the prmtop file when it loads a molecule. These values can be loaded relatively easily using the tkconsole, however. To do so:

1. select “Tk Console” from the “Extensions” menu in the “VMD Main” window.
    - The “VMD TkConsole” window will then open
2. Be sure that the atom you want to import radii for is the top molecule on the list in the VMD Main window. If it is not, you will need to replace “top” with the appropriate ID
3. Type or copy and paste the following lines, but DO NOT hit enter yet.

    ```
    set prot [atomselect “top” all]
    $prot set radius {#RadiiList#}
    ```

4. You will now need to replace #RadiiList# with the one from the prmtop file.
    a) Open the prmtop file for the molecule using a text editor
    b) find the section that starts with “%FLAG RADII”
    c) Highlight/Select the list of numbers that follows “%FORMAT(5E16.8)”
    d) Copy the list (usually done by selecting “Copy” from the “Edit” menu in your text editor)
    e) Go back to the “VMD TkConsole” window
    f) Highlight/Select #RadiiList#
    g) Select “Paste Ctrl-v” from the “Edit” menu in the “VMD TkConsole” window
5. Now hit return
    - If this was successful, you should now have the correct radii for each atom in the molecule.
    - you can have the console print the list of all radii by typing:

      ```
      $prot get radius
      ```

    - For a more human readable printout, use:

      ```
      for {set ind 0} {$ind<[llength $rad]} {incr ind} \
      {puts "Atom $ind radius is [lindex $rad $ind]"}
      ```

These radii are used by VMD to display the VDW surface (made by selecting “VDW” from the “Drawing Method” pull down menu in the “Graphical Representations” window). One useful trick is to set them to be a small amount larger (say .01 Å) than those used to generate the surface. This will ensure that the color map will represent the external field just outside of the molecule. To modify the radii type or copy the following in the Tk Console:

```
set rad [$prot get radius]
for {set ind 0} {$ind<[llength $rad]} {incr ind} \
{lset rad $ind [expr [lindex $rad $ind] +.01]}
```

The above code will increase all atomic radii by .01 angstroms. This can be changed if a different amount is desired. (The code assumes you already followed steps 1 through 5 otherwise $prot will be undefined!)

6.5. *pbsa* in *sander* and NAB

6.5.1. Electrostatic forces/gradients in *pbsa*

Force calculation in the finite-difference Poisson-Boltzmann method is straightforward, though not a trivial issue. It can be shown, by using the variation of the electrostatic free energy, that the electrostatic force density consists of three components, viz., the reaction field force, the dielectric boundary force, and the ionic force. [191] Since the ionic force is much smaller in absolute value than the other two components, we only include the reaction field force and the dielectric boundary force in this release.

The reaction field force only exists where there are atomic charges, so that it is straightforward to be mapped onto atoms. In contrast, the dielectric boundary force exists on the molecular surface where the dielectric constant changes. The surface force, or pressure, cannot be easily mapped onto atoms. This is because a force-mapping procedure from the molecular surface to atoms apparently needs the derivatives of molecular surface with respect to atomic positions. However such derivatives do not exist for the widely used molecular surface definition, i.e., the solvent excluded surface (SES). We are actively developing an analytical molecular surface definition that is consistent with the widely used SES definition for the numerical PB methods so that this difficulty will be overcome in future releases.

Temporarily, a partial solution in the mapping of dielectric boundary force as described by Gilson et al[191] is implemented for PB dynamics and minimization when the SES definition is used. The stability of the MD simulation has been much improved with a more accurate mapping method of analytical SES.

6.5.2. Example for *pbsa* in *sander*

All *pbsa* functionalities are available in *sander* and all input options are exactly the same as in the standalone *pbsa*. An apparent exception is IPB: you need to really set IPB to nonzero in order to invoke *pbsa* functionalities. All other default values of PB options in *sander* are same as those in *pbsa* for single point calculations, whereas there are some options that have different recommended or default values when PB minimization or dynamics is enabled. These options are

```
space=0.25
arcres=0.125
fscale=4
eneopt=2
bcopt=6
frcopt=2
```

The SPACE, ARCRES and FSCALE are all set for higher resolution of the grid so that the force calculation can be more accurate. The charge view method (ENEOPT = 2, FRCOPT = 2) is used here because it has been tested to be able to run stable molecular dynamics simulations. Plus, BCOPT is set to 6 to remove charge singularity for the same stability purpose. An example input for PBMD is given as follows

```
Sample PB visualization input
&cntrl
imin=0, ntx=1, irest=0,
ipb=2, ntb=0,
ntc=2, ntf=2,
tempi=100, temp0=100, ntt=3, gamma_ln=1,
nstlim=100000, dt=0.002,
ntpr=100, ntwr=100000, ntwx=100,
/
&pb
npbgrid=500, nsnba=5,
/
```

IPB is explicitly set to 2 to enable PB dynamics. The NPBGRID option is set to 500, which means the finite difference grid is regenerated every 500 dynamics steps. NSNBA = 5 means the atom-based pairlist is generated every 5 steps. Please refer to Chapter 17 for the other &cntrl options. Note that the above input can be used with *sander* only.

6.5.3. Example for *pbsa* in NAB

pbsa functionalities are available in NAB as a part of the standard build. However the available input options are limited, please refer to the table in Section 39.1 for the list of available *pbsa* input options. The structures and parameters are supplied by NAB's facility. Here is a sample of calls in a NAB program to the *mm_options()* routine, in order to run pbsa:

```
mm_options("ntpr=1, cut=99.0"); // No solute-solute cutoff
mm_options("ipb=2"); // Use PBSA
mm_options("accept=0.000001"); // Convergence criterion
mm_options("dprob=1.6"); // Solvent probe radius for SASA
mm_options("radiopt=1"); // Useatom-type/charge-based radii
mm_options("fillratio=4"); // Coarse/Fine ratio of electrostatic focusing
```

6.6. GPU accelerated *pbsa*

In this release of *pbsa*, one of the newer features added is the ability to use NVIDIA GPUs to accelerate implicit solvent PB simulations.[192] The GPU version of *pbsa* is called *pbsa.cuda*. At present several FDPB solvers have been ported to GPU using CUDA, while with the other *pbsa* routines unchanged. The default solver is set to Jacobi-preconditioned CG, which is the optimal one as shown in our testing. Our extensive analysis showed that a speedup ratio of about 5 to 50 can be achieved, depending on the size/structure of a given system. Note that the timing measurement was on the solver phase, i.e. the most computation demanding part in *pbsa* calculation, which means the overall simulation performance may not be that promising. While this GPU acceleration is considered to be production ready, it is still evolving and has not been tested to the same extent as the CPU code; users should excercise caution when using this code. The error checking on the GPU is not as verbose as it is on the CPU. In particular, simulation failures such as failed PB setup or other simulation instabilities, may manifest themselves as CUDA launch errors or GPU download failures. These are not informative error messages. If you encounter problems during a simulation on the GPU you should first try to run the identical simulation on the CPU to ensure that it is not your simulation setup causing the problems. Feedback and questions should be posted to the Amber mailing list (see *http://lists.ambermd.org/*). Future development of *pbsa.cuda* will aim to more robust code and user-friendly interface, and more performance-boost. At this time, however, we recommend expert users who have experiences in CUDA and *pbsa* calculations to use this GPU version.

This section of the manual describes the feature set, accuracy and memory considerations, installation and other aspects of *pbsa.cuda* at the time of Amber's release. Note that the rapidly changing nature of this field means the frequent updates are likely. You should refer to the AmberTools update page (see *http://ambermd.org/bugfixesat.html*) for the most up to date information.

6.6.1. Supported features

The GPU accelerated *pbsa* supports linear FDPB solvers such as CG and preconditioned-CG. The available option for this release is Jacobi-preconditioned CG, which has a better performance than other GPU solvers such as plain CG, ICCG, ILU and red-black SOR as shown in our analysis. To use this feature, the solver option of *pbsa.cuda* should be specified as:

```
solvopt=3
```

The use of this solver requires resetting MAXITN to a much larger number, i.e. 5000 (versus the default value of 100 for the default solver on the CPUs). This is to make sure the solver does not stop prematurely. In addition, both free boundary condition (FBC) and periodic boundary condition (PBC) are supported. The latter option is useful when simulating periodic systems such as those with membranes. The boundary condition options to use this feature are:

```
bcopt=5, or 1 (for FBC)
```

or

```
bcopt=10 (for PBC)
```

Within the use of SOLVOPT and BCOPT options as above, all other standard serial *pbsa* features will be supported as is; you should refer to the previous sections on the usage of serial *pbsa*.

An example input of single point slovation free energy calculation using *pbsa.cuda* is as follows:

```
&cntrl
  ntx=1, imin=1, ipb=1, inp=0
/
&pb
  npbverb=1, istrng=0, epsout=80.0, epsin=1.0, space=.5,
  accept=0.000001, dprob=1.4, radiopt=1, fillratio=1.5,
  smoothopt=0, arcres=0.0625, nfocus=1,
  bcopt=10, solvopt=3, maxitn=5000
/
```

6.6.2. Supported GPUs

pbsa.cuda has been implemented using CUDA and thus will only run on NVIDIA GPUs at present. Since the GPU code only uses single precision thus there is no requirement for GPU hardware to massively support double precison. In accordance with Amber CUDA requirements, compute capability 2.0 or above is required. However, be aware that the compute 2.0, sm 2.0, and sm 2.1 architectures are deprecated, and may be removed from NVIDIA *nvcc* in the future.

pbsa.cuda code development was on NVIDIA GeForce GTX 980 Ti and Intel Xeon E5-1620 v3 CPU. We also tested the released code on several other GPU platforms including laptop cards and obtained decent results. Though not having tested on all types of GPU cards, we suppose that most mid- to high-end cards should be supported.

Currently selection of which GPU is used for single GPU runs is automatic if the GPUs are set to process exclusive mode (`nvidia-smi -c 3`), but the recommended approach is to use the CUDA_VISIBLE_DEVICES environment variable to select which GPU should be used. More details are provided in the section 6.6.4.

6.6.3. Accuracy consideration and memory usage

Traditionally, *pbsa* code uses double precision. While for most consumer-grade GPU cards, single precision operations are widely supported with high efficiency. Double precision operations are possible, but are at a significant performance disadvantage. Therefore, in *pbsa.cuda* code we adopted a hybrid precision scheme: the calculation of electrostatic potential by CUDA solver uses single precision, while the system setup and the calculation of energy and force on CPU use double precision. Extensive tests of electrostatic solvation energy showed the correlation coefficients between hybrid and double precisions are 1.0 for both 10^{-3} and 10^{-6} convergence criteria. The maximum relative errors are 3.9×10^{-3} and 5.8×10^{-6}, respectively.

Memory usage is crucial for GPU implementations since memory is often limited on most consumer-grade graphics cards. In the default Jacobi-preconditioned CG implementation, typical GPU memory usage is about $184 \times Ngrid$ bytes, where $Ngrid$ is the number of grid nodes when discretizing the system with the finite difference method. You can refer to NVIDIA hardware manage tool *nvidia-smi* to obtain the runtime memroy allocation information. Our analysis of the Jacobi-preconditioned CG showed that it was able to successfully complete linear PBE calculations with $\sim$ 29.6 million grid nodes on the NVIDIA GTX 980 Ti cards, which have $\sim$ 6 GB GPU memory.

6.6.4. Installation and testing

pbsa.cuda must be built separately from the standard serial *pbsa* installation. Before attempting to build the GPU version of *pbsa* we recommend you have built and tested at least the serial version of Amber and AmberTools. This will help to ensure that basic issues relating to standard compilation on your hardware and operating system do not lead to confusion with GPU related compilation and testing problems. You should also be familiar with Amber's compilation and test procedures.

It is assumed that you have already correctly installed and tested CUDA support on your GPU. Additionally the environment variable CUDA_HOME should be set to point to your NVIDIA Toolkit installation and

$CUDA_HOME/bin/ should be in your $PATH. At the time of release, CUDA-7.5 or later CUDA-8.0 depending on your hardware architecture is required and is officially supported.

Building and testing the GPU code

To build *pbsa.cuda*, excute as follows in terminal:

```
export CUDA_HOME=/usr/local/cuda-8.0 (or other appropriate CUDA location)
cd $AMBERHOME
./configure -cuda gnu
make install
```

For intel and clang compilers, replace the ./configure step with the following commands:

```
./configure -cuda intel (for intel)
```

or

```
./configure -cuda clang (for clang)
```

Next you can run the tests using the default GPU with:

```
cd $AMBERHOME/AmberTools/test
make test.cuda
```

Note on some intel platforms, you need to use a larger stack size other than the system default setting to avoid stack overflow fails when running *pbsa.cuda*. The following command should do the trick:

```
ulimit -s unlimited
```

Testing alternative GPUs

If you wish to run the tests on a GPU different from the default one, then you can make use of the CUDA_VISIBLE_DEVICES environment variable to restrict the devices that *pbsa.cuda* sees as described below. For example, to test the GPU with ID = 1 (GPUs are numbered sequentially from zero) you can run the tests as follows:

```
cd $AMBERHOME/AmberTools/test
export CUDA_VISIBLE_DEVICES=1
make test.cuda
```

Or if you want to mask a specific GPU, use the following instead:

```
export CUDA_VISIBLE_DEVICES=0,2,3
```

In this way, devices 0, 2, 3 will be visible to *pbsa.cuda* and device 1 is masked. Note in this case devices 0, 2, 3 will be enumerated as devices 0, 1, 2.

To determine the device ID for the available hardware in your system, you can run NVIDIA's deviceQuery executable included in the CUDA SDK, after unsetting CUDA_VISIBLE_DEVICES environment variable:

```
unset CUDA_VISIBLE_DEVICES
deviceQuery
```

7. Reference Interaction Site Model

In addition to explicit and continuum implicit solvation models, Amber also has a third type of solvation model for molecular mechanics simulations, the reference interaction site model (RISM) of molecular solvation[193–206]. In AmberTools, 1D-RISM is available as rism1d. 3D-RISM is available as an option in NAB, MMPBSA.py and sander. rism3d.snglpnt is a simplified, standalone interface, ideal for calculating solvation thermodynamics on individual structures and trajectories. Details specific to using sander and sander.MPI can be found in Chapter 17.

7.1. Introduction

RISM is an inherently microscopic approach, calculating the equilibrium distribution of the solvent, from which all thermodynamic properties are then arrived at. Specifically, RISM is an approximate solution to the Ornstein-Zernike (OZ) equation[194, 203, 204, 207, 208]

$$h(r_{12},\Omega_1,\Omega_2) = c(r_{12},\Omega_1,\Omega_2) + \rho \int d\mathbf{r}_3\, d\Omega_3\, c(r_{13},\Omega_1,\Omega_3)\, h(r_{32},\Omega_3,\Omega_2), \tag{7.1}$$

where r_{12} is the separation between particles 1 and 2 while Ω_1 and Ω_2 are their orientations relative to the vector $\mathbf{r}_{12}$. The two functions in this relation are h, the total correlation function, and c, the direct correlation function. The total correlation function is defined as

$$h_{ab}(r_{ab},\Omega_a,\Omega_b) \equiv g_{ab}(r_{ab},\Omega_a,\Omega_b) - 1,$$

where g_{ab} is the pair-distribution function, which gives the conditional density distribution of species b about a. In cases where only radial separation is considered, for example by orientational averaging over site α of species a and site γ of species b, gives the familiar one dimensional site-site radial distribution function, $g_{\alpha\gamma}(r_{\alpha\gamma})$.

For real mixtures, it is often convenient to speak in terms of a solvent, V, of high concentration and a solute, U, of low concentration. A generic case of solvation is infinite dilution of the solute, i.e., $\rho^{\mathrm{U}} \to 0$. We can rewrite Equation (7.1), in the limit of infinite dilution, as a set of three equations:

$$h^{\mathrm{VV}}(r_{12},\Omega_1,\Omega_2) = c^{\mathrm{VV}}(r_{12},\Omega_1,\Omega_2) + \rho^{\mathrm{V}} \int d\mathbf{r}_3\, d\Omega_3\, c^{\mathrm{VV}}(r_{13},\Omega_1,\Omega_3)\, h^{\mathrm{VV}}(r_{32},\Omega_3,\Omega_2), \tag{7.2}$$

$$h^{\mathrm{UV}}(r_{12},\Omega_1,\Omega_2) = c^{\mathrm{UV}}(r_{12},\Omega_1,\Omega_2) + \rho^{\mathrm{V}} \int d\mathbf{r}_3\, d\Omega_3\, c^{\mathrm{UV}}(r_{13},\Omega_1,\Omega_3)\, h^{\mathrm{VV}}(r_{32},\Omega_3,\Omega_2), \tag{7.3}$$

$$h^{\mathrm{UU}}(r_{12},\Omega_1,\Omega_2) = c^{\mathrm{UU}}(r_{12},\Omega_1,\Omega_2) + \rho^{\mathrm{V}} \int d\mathbf{r}_3\, d\Omega_3\, c^{\mathrm{UV}}(r_{13},\Omega_1,\Omega_3)\, h^{\mathrm{VU}}(r_{32},\Omega_3,\Omega_2). \tag{7.4}$$

Equation (7.3) is directly relevant for biomolecular simulations where we are often interested in the properties of a single, arbitrarily complex solute in the solution phase. Solutions to Equation (7.3) can be obtained using 3D-RISM. However, a solution to Equation (7.2) for pure solvent is a necessary prerequisite and is readily obtained from 1D-RISM.

To obtain a solution to the OZ equations it is necessary to have a second equation that relates h and c or uniquely defines one of these functions. The general closure relation is[207]

$$g(r_{12},\Omega_1,\Omega_2) = \exp\left[-\beta u(r_{12},\Omega_1,\Omega_2) + h(r_{12},\Omega_1,\Omega_2) - c(r_{12},\Omega_1,\Omega_2) + b(r_{12},\Omega_1,\Omega_2)\right] \tag{7.5}$$

u is the potential energy function for the two particles and b is known as the bridge function (a non-local functional, representable as infinite diagrammatic series in terms of h [207]). It should be noted that u is the only point at which the interaction potential enters the equations. Depending on the method used to solve the OZ equations, u

is generally an explicit potential. In principle, it should now be possible to solve our two equations. For example, we may wish to use SPC/E as a water model. Inputting the relevant aspects of the SPC/E model into u, 1D-RISM can be used to calculate the equilibrium properties of the SPC/E model. A different explicit water model will yield different properties.

A fundamental problem for all OZ-like integral equation theories is the bridge function, which contains multiple integrals that are readily solved only in special circumstances. In practice, an approximate closure relation must be used. While many closures have been developed, at this time only three are implemented in 3D-RISM: hypernetted-chain approximation (HNC), Kovalenko-Hirata (KH) and the partial series expansion of order-n (PSE-n).

For HNC, we set $b = 0$, giving[207]

$$\begin{aligned} g^{\mathrm{HNC}}(r_{12},\Omega_1,\Omega_2) &= \exp\left(-\beta u(r_{12},\Omega_1,\Omega_2) + h(r_{12},\Omega_1,\Omega_2) - c(r_{12},\Omega_1,\Omega_2)\right) \\ &= \exp\left(t^*(r_{12},\Omega_1,\Omega_2)\right) \end{aligned} \tag{7.6}$$

where t^* is the renormalize-indirect correlation function. HNC works well in many situations, including charged particles, but has difficulties when the size ratios of particles in the system are highly varied and may not always converge on a solution when one should exist. Also, as the bridge term is generally repulsive, HNC allows particles to approach too closely, overestimating non-Coulombic interactions[204].

KH is a combination of HNC and the mean spherical approximation (MSA), the former being applied to the spatial regions of solvent density depletion ($g < 1$), including the repulsive core, and the latter to those of solvent density enrichment ($g > 1$), such as association peaks[203, 204]

$$g^{\mathrm{KH}}(r_{12},\Omega_1,\Omega_2) = \begin{cases} \exp\Big(t^*(r_{12},\Omega_1,\Omega_2)\Big) & \text{for } g(r_{12},\Omega_1,\Omega_2) \le 1 \\ 1 + t^*(r_{12},\Omega_1,\Omega_2) & \text{for } g(r_{12},\Omega_1,\Omega_2) > 1 \end{cases}. \tag{7.7}$$

Like HNC, KH handles Coulombic systems well but overestimates non-Coulombic interactions. Unlike HNC, it does not have difficulties with highly asymmetric particle sizes and readily converges to stable solutions for almost all systems of practical interest. The reliability of the KH closure makes it particularly suitable for molecular mechanics calculations.

PSE-n offers the ability to interpolate between KH and HNC. Here, the exponential regions of solvent density enrichment are treated as a Taylor expansion,

$$g^{\text{PSE-n}}(r_{12},\Omega_1,\Omega_2) = \begin{cases} \exp\Big(t^*(r_{12},\Omega_1,\Omega_2)\Big) & \text{for } g(r_{12},\Omega_1,\Omega_2) \le 1 \\ \sum_{i=0}^{n} {}^{(t^*(r_{12},\Omega_1,\Omega_2))^i}/_{i!} & \text{for } g(r_{12},\Omega_1,\Omega_2) > 1 \end{cases}. \tag{7.8}$$

In the case of $n = 1$, the KH closure is obtained, while in the limit of $n \to \infty$ HNC is recovered. This allows a balance between the numerical stability of KH and the often better accuracy of HNC.

7.1.1. 1D-RISM

1D-RISM is used to calculate bulk properties of the solvent and is a prerequisite for 3D-RISM, for which the primary result is the bulk solvent site-site susceptibility in reciprocal space, $\chi^{\mathrm{VV}}(k)$. As its name would suggest, 1D-RISM is a one-dimensional calculation. The six-dimensional OZ equations are reduced to one dimension (radial separation) via the fundamental RISM approximation[194–197, 207, 208], which produces the intramolecular pair correlation matrix,

$$\omega_{\alpha\gamma}(k) = \sin(kr_{\alpha\gamma})/(kr_{\alpha\gamma}) \tag{7.9}$$

where α and γ label the different atom types in the model. Note that atoms of the same type in RISM theory have the same Lennard-Jones and Coulomb parameters. For example, most three site water models have two RISM types, oxygen and hydrogen. Depending on the model, propane, C_3H_8, may have two carbon types and two hydrogen types. Equation (7.2) then becomes

$$\begin{aligned} h_{\alpha\gamma}(r) &= \sum_{\mu\nu} \int d\mathbf{r}' d\mathbf{r}'' \omega_{\alpha\mu}(\left|r-r'\right|) c_{\mu\nu}(\left|r'-r''\right|) \left[\omega_{\nu\gamma}(r'') + \rho_\nu h_{\nu\gamma}(r'')\right] \\ &= \frac{1}{(2\pi)^3} \int e^{i\mathbf{k}\cdot\mathbf{r}} d\mathbf{k} \left[\boldsymbol{\omega}\mathbf{c}\left[\mathbf{1} - \rho\boldsymbol{\omega}\mathbf{c}\right]^{-1}\boldsymbol{\omega}\right]_{\alpha\gamma} \\ &= \sum_0^\infty \boldsymbol{\omega}(k)\mathbf{c}(k)\boldsymbol{\omega}(k)\left[\rho\mathbf{c}(k)\boldsymbol{\omega}(k)\right]^n . \end{aligned} \tag{7.10}$$

Equation (7.10) must be complemented with one of the five closures currently supported by `rism1d` (see Subsection 7.4.1). In 1d, these are site-site closures and there is no orientational dependence. For example, the HNC closure (Eq. (7.6)) becomes,

$$g_{\alpha\gamma}^{\mathrm{HNC}}(r) = \exp\left[-\beta u_{\alpha\gamma}(r) + h_{\alpha\gamma}(r) - c_{\alpha\gamma}(r)\right]. \tag{7.11}$$

Equation (7.10), with KH, HNC or PSE-n closures, is readily applicable to liquid mixtures, with site indices of the site-site correlation functions enumerating interaction sites on all (different) species in the solution and the intramolecular matrix (7.9) set equal to zero for sites α,γ belonging to different species.

A dielectrically consistent version of 1D-RISM theory (DRISM) enforces the proper dielectric asymptotics of the site-site correlation functions, and so provides the self-consistent dielectric properties of electrolyte solution with polar solvent and salt in a range of concentrations, including the given dielectric constant of the solution [209].

The 1D-RISM integral equations are then solved for the site-site direct correlation function in an iterative manner, accelerated by the modified direct inversion of the iterative subspace (MDIIS) [204, 210]. All correlation functions are represented as one-dimensional grids and the convolution integrals in Equation (7.10) are performed in reciprocal space by making use of a fast Fourier transform applied to the short-range parts of all the correlations, while the electrostatic asymptotics are separated out and Fourier transformed analytically [204–206].

1D-RISM is a general method and not restricted to water or pure solvents. For example, 1D-RISM may be used to treat solutions of aqueous alkali and halide ions at various concentrations [211]. The output from 1D-RISM can then be used for complex solutes, such as DNA [212], in 3D-RISM.

7.1.2. 3D-RISM

With the results from 1D-RISM, a 3D-RISM calculation for a specific solute can be carried out. For 3D-RISM calculations, only the solvent orientational degrees of freedom are averaged over and Equation (7.3) becomes[202, 203]

$$h_\gamma^{\mathrm{UV}}(\mathbf{r}) = \sum_\alpha \int d\mathbf{r}' c_\alpha^{\mathrm{UV}}(\mathbf{r}-\mathbf{r}')\, \chi_{\alpha\gamma}^{\mathrm{VV}}(r'), \tag{7.12}$$

where $\chi_{\alpha\gamma}^{\mathrm{VV}}(r)$ is the site-site susceptibility of the solvent, obtained from 1D-RISM and given by

$$\chi_{\alpha\gamma}^{\mathrm{VV}}(r) = \omega_{\alpha\gamma}^{\mathrm{VV}}(r) + \rho_\alpha h_{\alpha\gamma}^{\mathrm{VV}}(r).$$

3D-RISM supports HNC, KH and PSE-n closures (see Sections 7.6.1, 39.1 and 30.3.1). As with the 1D-RISM closures, these are constructed by analogy from Eqs. 7.6-7.8. For example, HNC becomes

$$g_\gamma^{\mathrm{HNC,UV}}(\mathbf{r}) = \exp\Big(-\beta u_\gamma^{\mathrm{UV}}(\mathbf{r}) + h_\gamma^{\mathrm{UV}}(\mathbf{r}) - c_\gamma^{\mathrm{UV}}(\mathbf{r})\Big). \tag{7.13}$$

As with 1D-RISM, correlation functions are represented on (3D) grids, convolution integrals are performed in reciprocal space and a self-consistent solution is iteratively converged upon using the MDIIS accelerated solver. There is one 3D grid for each solvent type for each correlation function. For example, for a solute in SPC/E water there will be both $g_{\mathrm{H}}^{\mathrm{UV}}(\mathbf{r})$ and $g_{\mathrm{O}}^{\mathrm{UV}}(\mathbf{r})$ grids. Each point on the $g_{\mathrm{H}}^{\mathrm{UV}}(\mathbf{r})$ will give the fractional density of water hydrogen a that location of real-space.

To properly treat electrostatic forces in electrolyte solution with polar molecular solvent and ionic species, the electrostatic asymptotics of all the correlation functions (both the 3D and radial ones) are treated analytically [204, 205, 213]. The non-periodic electrostatic asymptotics are separated out in the direct and reciprocal space and the remaining short-range terms of the correlation functions are discretized on a 3D grid in a non-periodic box large enough to ensure decay of the short-range terms at the box boundaries [213]. The convolution of the short-range terms in the integral equation (7.12) is calculated using 3D fast Fourier transform [214, 215]. Accordingly, the electrostatic asymptotics terms in the thermodynamics integral (7.15) below are handled analytically and reduced to one-dimensional integrals easy to compute [213].

With a converged 3D-RISM solution for h^{UV} and c^{UV}, it is straightforward to calculate solvation thermodynamics. From the perspective of molecular simulations, the most important thermodynamic values are the excess chemical potential of solvation (solvation free energy), μ^{ex} and the mean solvation force, $\mathbf{f}_i^{\mathrm{UV}}(\mathbf{R}_i)$, on each solute atom, i. μ^{ex} can be obtained through analytical thermodynamic integration for HNC,

$$\mu^{\mathrm{ex,HNC}} = k_{\mathrm{B}}T\sum_{\alpha}\rho_{\alpha}^{\mathrm{V}}\int d\mathbf{r}\left[\frac{1}{2}\left(h_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right)^2 - c_{\alpha}^{\mathrm{UV}}(\mathbf{r}) - \frac{1}{2}h_{\alpha}^{\mathrm{UV}}(\mathbf{r})c_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right], \tag{7.14}$$

KH ,

$$\mu^{\mathrm{ex,KH}} = k_{\mathrm{B}}T\sum_{\alpha}\rho_{\alpha}^{\mathrm{V}}\int d\mathbf{r}\left[\frac{1}{2}\left(h_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right)^2\Theta\left(-h_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right) - c_{\alpha}^{\mathrm{UV}}(\mathbf{r}) - \frac{1}{2}h_{\alpha}^{\mathrm{UV}}(\mathbf{r})c_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right], \tag{7.15}$$

and PSE-n,

$$\begin{aligned}\mu^{\mathrm{ex,PSE}\text{-}n} = k_{\mathrm{B}}T\sum_{\alpha}\rho_{\alpha}^{\mathrm{V}}\int d\mathbf{r}\Bigg[&\frac{1}{2}\left(h_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right)^2 - c_{\alpha}^{\mathrm{UV}}(\mathbf{r}) - \frac{1}{2}h_{\alpha}^{\mathrm{UV}}(\mathbf{r})c_{\alpha}^{\mathrm{UV}}(\mathbf{r}) \\ &- \frac{\left(t^*(\mathbf{r})\right)^{n+1}}{(n+1)!}\Theta\left(h_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right)\Bigg],\end{aligned} \tag{7.16}$$

where Θ is the Heaviside function.

Analogous versions of Eqns. 7.6, 7.15 and 7.16 are used in 1D-RISM. While these are used for DRISM they are have been derived for XRISM. Furthermore, these equations have been derived a number of different ways with slightly different functional forms of the $-\frac{1}{2}hc$ term [203, 216–219]. These different functional forms are equivalent in XRISM but not in DRISM. The form introduced by Pettitt and Rossky [217] is the most popular in the literature and the default selection in `rism1d`. It is possible to have `rism1d` evaluate and output all three functional forms (see Output) but, for DRISM, none of these expressions are strictly correct.

The force equation

$$\mathbf{f}_i^{\mathrm{UV}}(\mathbf{R}_i) = -\frac{\partial\mu^{\mathrm{ex}}}{\partial\mathbf{R}_i} = -\sum_{\alpha}\rho_{\alpha}\int d\mathbf{r}g_{\alpha}^{\mathrm{UV}}(\mathbf{r})\frac{\partial u_{\alpha}^{\mathrm{UV}}(\mathbf{r}-\mathbf{R}_i)}{\partial\mathbf{R}_i}$$

is valid for all closures with a path independent expression for the excess chemical potential, such as HNC, KH and PSE-n closures implemented in 3D-RISM [193, 220–222].

In addition to closure specific expressions for the solvation free energy, other approximations also exist. The Gaussian fluctuation (GF) approximation[223, 224] is given as

$$\mu^{\mathrm{ex,GF}} = k_{\mathrm{B}}T\sum_{\alpha}\rho_{\alpha}^{\mathrm{V}}\int d\mathbf{r}\left[-c_{\alpha}^{\mathrm{UV}}(\mathbf{r}) - \frac{1}{2}h_{\alpha}^{\mathrm{UV}}(\mathbf{r})\,c_{\alpha}^{\mathrm{UV}}(\mathbf{r})\right] \tag{7.17}$$

and has been shown to yield improved absolute solvation free energies for both polar and non-polar solutes[224, 225] but not necessarily for relative free energies[226]. It is not associated with a particular closure but is typically used in place of the expression for a given closure.

Eqs. (7.14)-(7.16) give the total solvation free energy, ΔG_{sol}, but it is often useful to decompose this into electrostatic (solvent polarization), ΔG_{pol}, and non-electrostatic (dispersion and cavity formation), $(\Delta G_{\mathrm{dis}} + \Delta G_{\mathrm{cav}})$, terms. Conceptually, we can divide the path of the thermodynamic integration into two steps: first the solute without partial charges is inserted into the solvent (dispersion and cavity formation) and then partial charges are introduced, which polarize the solvent,

$$\mu^{\mathrm{ex}} = \Delta G_{\mathrm{sol}} = \Delta G_{\mathrm{pol}} + \Delta G_{\mathrm{dis}} + \Delta G_{\mathrm{cav}}.$$

ΔG_{sol} is produced by a 3D-RISM calculation on the charged solute. ΔG_{pol} is then the difference of the two calculations. As a point of reference, generalized-Born and Poisson-Boltzmann methods calculate only ΔG_{pol} and, typically, use a calculation involving solvent accessible surface area to predict $\Delta G_{\mathrm{dis}} + \Delta G_{\mathrm{cav}}$.

7.1.3. Analytic Temperature Derivatives

For the thermodynamic analysis of solvation, it is often useful to calculate the energetic and entropic contributions, $\varepsilon^{solv}$and $-TS^{solv}$ respectively, to the solvation free energy. It has been shown that it is possible to analytically decompose the solvation free energy into these two contributions when the solvation free energy has a closed analytical form, such as with HNC and KH closure [227]. In what follows, the analytical expression of energetic and entropic contributions to the solvation free energy are derived in the framework of 1D-RISM theory with HNC closure. The similar derivation can be applied to other closures as well as to the framework of 3D-RISM theory. At this time, temperature derivatives are implemented for with HNC, KH and PSE-*n* closures in both 1D- and 3D-RISM [228].

The solvation free energy of species U in a solution consisting of N total species is expressed in the RISM-HNC framework as

$$\mu_{\mathrm{HNC}}^{\mathrm{ex,U}} = k_{\mathrm{B}}T\sum_{\alpha}^{\mathrm{on\ U}}\sum_{M=1}^{N}\sum_{\gamma}^{\mathrm{on}\ M}\rho_\gamma\int d\mathbf{r}\left[\tfrac{1}{2}\left(h_{\alpha\gamma}(r)\right)^2 - c_{\alpha\gamma}(r) - \tfrac{1}{2}h_{\alpha\gamma}(r)c_{\alpha\gamma}(r)\right].$$

The differentiation of the solvation free energy with respect to the temperature T leads to

$$\delta_T\mu_{\mathrm{HNC}}^{\mathrm{ex,U}} =$$

$$\mu_{\mathrm{HNC}}^{\mathrm{ex,U}} + k_{\mathrm{B}}T\sum_{\alpha}^{\mathrm{on\ U}}\sum_{M=1}^{N}\sum_{\gamma}^{\mathrm{on}\ M}\rho_\gamma\int d\mathbf{r}\left[h_{\alpha\gamma}(r)\cdot\delta_T h_{\alpha\gamma}(r) - \delta_T c_{\alpha\gamma}(r) - \tfrac{1}{2}\delta_T h_{\alpha\gamma}(r)\cdot c_{\alpha\gamma}(r) - \tfrac{1}{2}h_{\alpha\gamma}(r)\cdot\delta_T c_{\alpha\gamma}(r)\right].$$

where δ_T is $T\frac{\partial}{\partial T}$. Since $\mu_{\mathrm{HNC}}^{\mathrm{ex,U}} = \varepsilon^{solv,U} - TS^{solv,U}$, we have $\delta_T\mu_{\mathrm{HNC}}^{\mathrm{ex,U}} = -TS^{solv,U}$ and therefore the above equation can be rearranged as

$$\varepsilon^{solv,U} =$$

$$-k_{\mathrm{B}}T\sum_{\alpha}^{\mathrm{on\ U}}\sum_{M=1}^{N}\sum_{\gamma}^{\mathrm{on}\ M}\rho_\gamma\int d\mathbf{r}\left[h_{\alpha\gamma}(r)\cdot\delta_T h_{\alpha\gamma}(r) - \delta_T c_{\alpha\gamma}(r) - \tfrac{1}{2}\delta_T h_{\alpha\gamma}(r)\cdot c_{\alpha\gamma}(r) - \tfrac{1}{2}h_{\alpha\gamma}(r)\cdot\delta_T c_{\alpha\gamma}(r)\right].$$

It is noted that the solvation energy $\varepsilon^{solv,U}$ can be viewed as consisting of two contributions: one arising from creation of a polarized cavity (in pure solvent) and the other corresponding to the energy of embedding the solute molecule into the cavity. The former is the solvent reorganization energy and the latter is the average solute-solvent interaction energy that is obtained as $\sum_\alpha\sum_\gamma\rho_\gamma\int d\mathbf{r}u_{\alpha\gamma}g_{\alpha\gamma}$.

The temperature derivatives of correlation functions$\delta_T h(r)$ and $\delta_T c(r)$ can be obtained by solving the temperature derivative of RISM-HNC equations

$$\delta_T\mathbf{h}(k) = \mathbf{w}(k)\delta_T\mathbf{c}(k)\mathbf{w}(k) + \rho\mathbf{w}(k)\delta_T\mathbf{c}(k)\mathbf{h}(k) + \rho\mathbf{w}(k)\mathbf{c}(k)\delta_T\mathbf{h}(k)$$

and

$$\delta_T h_{\alpha\gamma}(r) = \left[\frac{u_{\alpha\gamma}(r)}{k_B T} + \delta_T h_{\alpha\gamma}(r) - \delta_T c_{\alpha\gamma}(r)\right]\left(h_{\alpha\gamma}(r) + 1\right).$$

Some practical examples can be found in [229], [230] and [228].

7.2. Practical Considerations

7.2.1. Computational Requirements and Parallel Scaling

Calculating a 3D-RISM solution for a single solute conformation typically requires about 100 times more computer time than the same calculation with explicit solvent or PB. While there are other factors to consider, such as sampling confined solvent or overall efficiency of sampling in the whole statistical ensemble at once, this can be prohibitive for many applications. Memory is also an issue as the 3D correlation grids require anywhere from a few megabytes for the smallest solutes to gigabytes for large complexes. A lower bound and very good estimate for the total memory required is

$$\text{Total memory} \geq 8\,\text{bytes} \times \left[N_{\text{box}}\, N^{\text{V}} \left(\underbrace{2N_{\text{MDIIS}}}_{c,\text{residual}} + \underbrace{1}_{u} + \underbrace{N_{\text{decomp}}}_{\text{polar decomp}} \underbrace{N_{\text{propagate}}}_{\text{past solutions}} \right) \right.$$
$$\left. (N_{\text{box}} + 2N_yN_z) \left\{ \underbrace{4}_{\text{asymptotics}} + \underbrace{1}_{\text{FFT scratch}} + \underbrace{2}_{g,h} N^{\text{V}} \right\} \right]$$

where $N_{\text{box}} = N_x \times N_y \times N_z$ is the total number of grid points, N^{V} is the number of solvent atom species and N_{MDIIS} is the number of MDIIS vectors used to accelerate convergence. u^{UV}, c^{UV} and the residual of c^{UV} are stored in real-space only and require a full grid for each solvent. c^{UV} and its residual also require N_{MDIIS} grids for the MDIIS routine (see the mdiis_nvec keyword) and $N_{\text{propagate}}$ grids to make use of solutions from previous solute configurations to improve the initial guess (see the npropagate keyword). If a polar/non-polar decomposition is requested (see the polardecomp keyword) an additional set of grids for past solutions with no solute charges is kept ($N_{\text{decomp}} = 2$); by default this is turned off ($N_{\text{decomp}} = 1$). The full real space grid plus an additional $2N_yN_x$ grid points are needed (due to the FFT) for g and h for each solvent species and for the four grids required to compute the long range asymptotics. Memory, therefore, scales linearly with N_{box} while computation time scales as $O(N_{\text{box}} \log(N_{\text{box}}))$ due to the requirements of calculating the 3D fast Fourier transform (3D-FFT). To overcome these requirements, two options are available beyond optimizations already in place, multiple time steps and parallelization. Multiple time step methods are available only in sander (Chapter 17) and are applicable to molecular dynamics calculations only. Parallelization is available for all calculations but is limited by system size and computational resources.

Both `sander` and `NAB` have MPI implementations of 3D-RISM (see Section 7.5.5 for NAB compiling instructions) that distribute both memory requirements and computational load. As memory is distributed, the aggregate memory of many computers can be used to perform calculations on very large systems. Memory distribution is handled by the FFTW 3.3 library so decomposition is done along the z-axis. If a variable solvation box size is used, the only consideration is to avoid specifying a large, prime number of processes (≥ 7). For fixed box sizes, the number of grids points in each dimension must be divisible by two (a general requirement) and the number of grid points in the z-axis must be divisible by the number of processes. `sander.MPI` also has the additional consideration that the number of processes cannot be larger than the number of solute residues; `NAB` does not suffer from this limitation.

7.2.2. Output

g^{UV}, h^{UV} and c^{UV} files can be output for 3D-RISM calculations and are useful for visualization and calculation of thermodynamic quantities. These use the ASCII Data Explorer (DX) file format (See http://ambermd.org/formats.html) so there is one file for each solvent atom type for each requested frame. Each file is $\left(348 + N_{\text{box}} \times 16\frac{1}{3}\right)$ bytes, which can quickly fill disk space. Also, very few visualization programs are capable of displaying both molecular and volumetric trajectories.

7.2.3. Numerical Accuracy

Numerical accuracy depends on the specified residual tolerance for the solution and the solvation box physical size and grid spacing. Almost all applications should use a grid spacing of 0.5 Å. A larger grid spacing quickly leads to severe errors in thermodynamic quantities. Smaller grid spacing may be necessary for some applications (e.g., mapping potentials of mean force) but this is rare and computationally expensive. A buffer distance between the solute and the edges of the solvent box should typically be at least 14 Å for water or larger for ionic solutions. The solvation box size should be increased until the thermodynamic properties converge (see Section 7.5.1). Molecular dynamics[193], minimization and trajectory post-processing[226] have different requirements for the maximum residual tolerance. Molecular dynamics does well with a tolerance of 10^{-5} and `npropagate=5`. Minimization requires tolerances of 10^{-11} or lower and is typically limited to `drms` $\geq 10^{-4}$. Trajectory post-processing for MM/RISM type calculations typically have high statistical noise from the trajectory itself and it is possible to use a tolerance of 10^{-3} and `npropagate=`1. However, this should be compared against a tolerance of 10^{-5} on a subset of the data before committing to this level of accuracy.

7.2.4. Solvation Free Energy Corrections

3D-RISM with HNC-like closures is known to overestimate the non-polar component of the solvation free energy. Several alternate expressions for the solvation free energy have been developed to correct this and are based all, or in part, on the partial molar volume (PMV) of the solute. These include the Universal Correction (UC) [231], Ng Bridge Correction (NgB) [232] and the Pressure Correction Plus (PC+/3D-RISM) correction [233]. 3D-RISM currently implements UC and PV+/3D-RISM as runtime options. NgB results can be calculated from the standard thermodynamic output if the polarDecomp option is used but is not implemented directly. UC and NgB are both parameterized corrections. So, parameters for these corrections must be used only with the .xvv file used to create them. Our implementation of UC uses the excess chemical potential of the closure rather than the GF functional, as we have found this provides better results in general [228]. All of these corrections have been almost exclusively used with pure water under ambient conditions, though there are promising results for UC with non-polar liquids.[234] Using these methods with different solvents and co-solvents is a subject of on-going research.

7.3. Work Flow

Using 3D-RISM with SANDER or NAB for molecular dynamics, minimization or snapshot analysis is very similar to using implicit solvent models like GBSA or PBSA. However, some additional preliminary setup is required, the extent of which depends on the solvent to be used.

3D-RISM requires detailed information of the bulk solvent in the form of the site-site susceptibility, χ^{VV}, and properties such as the temperature and partial charges. This is read in as an `.xvv` file, which is produced by a 1D-RISM calculation. If another 3D-RISM calculation is to be preformed with any details of the bulk solvent changed (e.g., temperature or pressure) a new `.xvv` file must be produced. Examples of precomputed `.xvv` files for SPC/E and TIP3P water can be found in `$AMBERHOME/AmberTools/test/rism1d`.

Special care must be taken when producing .xvv files for use with 3D-RISM, particularly with respect to grid parameters. It is important that the spatial extent of the grid be large enough to capture the essential long range features of the solvent while the spacing must be fine enough to sample the short-range structure. A grid spacing of 0.025 Å is sufficient for most applications. The number of grid points required, which will determine the physical length of the grid in Å, generally depends on the properties of the solvent. Low concentration aqueous salt solutions typically require much larger grids than pure bulk water. A good indicator that the grid is large enough is convergence of delhv0 in the .xvv file. When converged, delhv0 should retain four to five digits of precision when the number of grid points is doubled.

The ability of 3D-RISM to perform temperature derivatives and calculate solvation energy and entropy requires `.xvv` files with with temperature dependence information. `rism1d` must be run with entropicDecomp option turned on (Section 7.4.1). The version number in the `.xvv` file header indicates the maximum information available. Version 1.001 (current) allows temperature derivatives and solvation entropies and energies for all reported quantities.

Version 1.000 (since Amber12) does not allow temperature derivatives of the PMV or solventation energies and entropies of PMV-based corrections. Version 0.001 does not have information for any temperature derivatives.

1D-RISM calculations require details of the some bulk properties of the solvent, such as temperature and dielectric constant, and an explicit model of the molecular components. These are read in from one or more `.mdl` files, depending on the composition of the solvent. Several `.mdl` files are included in the Amber11 distribution and can be found in `$AMBERHOME/dat/rism1d/model`. These include many of the explicit models for solvent and ions used with the Amber force fields. Other solvents models may be used by creating appropriate MDL files. See Section 7.8 for format details.

7.3.1. Solution Convergence

The default parameters for 3D-RISM are selected to provide the best performance for the majority of systems. In cases where a convergence is not achieved, the strategies below may be useful.

7.3.1.1. Closure Bootstrap

When a PSE-*n* or HNC closure is desired, the most effective method to overcome convergence issues is to use a low order closure solution as a starting guess. The KH closure should be the starting point as it is numerically robust and, typically, converges easily in the vast majority of case. After this, higher orders of PSE-*n* can be used until the desired closure is reached. The procedure for 1D-RISM and 3D-RISM differs slightly in practice.

1D-RISM `rism1d` can use restart files to implement this approach (see Section Subsection 7.4.1). First, run `rism1d` with the KH closure to convergence. Then use the .sav file as input for the next highest closure. The root name of the .sav file must be the same as your .inp file. To avoid overwriting lower order solutions, name the files by closure or use separate directories. You will have to rename the .sav files as you go.

3D-RISM All 3D-RISM interfaces have closure bootstrapping builtin via the *closure* and *tolerance* keywords. Closures should be specified as an ordered list with last closure being the highest order closure. The solutions of the intermediate closures can have a high tolerance. The default tolerance for intermediate closures is 1 and there is no observed benefit to tolerances less than 1e-2. See details in Subsection 7.6.1, Subsection 7.7.2.1 and Section 39.1.

7.3.1.2. MDIIS Settings

MDIIS default setting are appropriate for most cases. Should your residual diverge or the solver get stuck on a particular value, you can try modest adjustments.

Decrease *mdiis_del* *mdiis_del* controls the step size of MDIIS. A smaller step size can help convergence but if this is set too small it can cause convergence problems. For `rism1d`, this should be no lower than 0.1 or 0.2. For 3D-RISM, it should be 0.5 at the lowest.

Increase *mdiis_nvec* This is the number of trial solutions that are saved for predicting a new solution. The optimal number for rapid convergence is typically 10 for 3D-RISM and 20 for 1D-RISM. However, for 3D-RISM, the default choice of 5 requires much less memory and is computationally faster even though more iterations are required. Increasing the *mdiis_nvec* may help for 3D-RISM but is unlikely to help for 1D-RISM.

Increase *mdiis_restart* Occasionally, the MDIIS routine goes in the wrong direction and the residual increases significantly. If it increases more than *mdiis_restart* then the MDIIS routine selects the solution with the lowest residual and purges the other trial solutions. The default value of 10 can be too aggressive and cause the solver to cycle. Increasing the value to 100 or 1000 sometimes allows the solver to recover from a misstep.

7.3.1.3. Parameter Annealing

Chargeless, hot gases are the easiest systems to converge. For 1D-RISM, this can be used to bootstrap a solution in a similar manner to closure bootstrapping. By slowly turning on charges, lowering the temperature or increasing the density, a converged solution may be reached. This only works for 1D-RISM because it requires restarting from a previous solution. As with closure bootstrapping, files should be carefully renamed during the procedure. There is no general protocol but the parameter increment should be reduced as the target value is approached. E.g., turning on charges in a linear fashion usually isn't helpful.

7.3.1.4. Forcefield selection

The forcefield may affect convergence due to the number of solvent sites involved or the particular parameters of the forcefield.

Number of Sites Molecules with more sites are more difficult to converge. Six or more sites is already difficult to converge and more that 10 may not be possible under any circumstances. One solution is to use a united atom or coarse grained forcefields to reduce the number of sites.

Alternate Parameterization Some parameter sets simply yield a stiffer set of equations to solve. Choosing an alternate parameter set may allow convergence with only small differences in the numerical results. For example, the cSPC/E water model with SPC/E Joung/Cheatham ions is easier to converge at higher ion concentrations in 1D-RISM than cTIP3P water with TIP3P Joung/Cheatham ions. Both models give nearly identical results in RISM at lower concentrations but NaCl in cTIP3P water will not converge above 0.5 M for the PSE-3 closure despite using all of the above methods.

7.4. rism1d

1D-RISM calculations are carried out with `rism1d`, and require only one input file with an `.inp` suffix. The input file is listed on the command line without this suffix.

```
rism1d inputfile
```

Parameters for the calculation are read in from `parameters` name list.

7.4.1. Parameters

Note that these keywords are not case sensitive.

Theory

theory [DRISM] The 1D-RISM theory to use.

DRISM Dielectrically consistent RISM (recommended).

XRISM Extended RISM.

closure [KH] The type of closure to use.

KH Kovalenko-Hirata (recommended).

PSE*n* Partial serial expansion of order *n*. E.g., “PSE3”.

HNC Hyper-netted chain equation.

PY Percus-Yevick.

entropicDecomp [1] Solve another set of integral equations to calculate the temperature derivative. This typically adds less than 50% to the compute time and yields an energy/entropy decomposition of the excess chemical potential for all species and sites.

0 Do not calculate the temperature derivative.

1 Calculate the temperature derivative.

Grid Size

dr [0.025] Grid spacing in real space in Å.

nr [16384] Number of grid points. Should be a product of small prime factors (2, 3 and 5).

Output

outlist [] Indicates what output files to produce. Output file names use the root name of the input file with an extension listed below. This is a list of any combination of the following characters in any order, upper or lower case.

- **U** $U^{\mathrm{VV}}(r)$ Solvent site-site potential in real space, `inputfile`.uvv (see http://ambermd.org/formats.html).
- **X** $\chi^{\mathrm{VV}}(k)$ Solvent site-site susceptibility in reciprocal space. Required input for 3D-RISM, `inputfile`.xvv (see http://ambermd.org/formats.html).
- **G** $G^{\mathrm{VV}}(r)$ Solvent site-site pair distribution function in real-space, `inputfile`.gvv (see http://ambermd.org/formats.htm
- **B** $B^{\mathrm{VV}}(r)$ Solvent site-site bridge correction in real space, `inputfile`.bvv (see http://ambermd.org/formats.html).
- **T** Thermodynamic properties of the solvent, `inputfile`.therm (see http://ambermd.org/formats.html).
- **E** ex$N^{\mathrm{VV}}(r)$, exN^{VV} Solvent site-site running, `inputfile`.exnvv, and total, `inputfile`.n00 (see http://ambermd.org/formats.html), excess coordination numbers in real space.
- **N** $N^{\mathrm{VV}}(r)$ Solvent site-site running coordination numbers in real space, `inputfile`.nvv (see http://ambermd.org/formats
- **Q** exQ^{VV} Solvent site-site excess total charge of site γ about α, `inputfile`.q00 (see http://ambermd.org/formats.html).
- **S** $S^{\mathrm{VV}}(k)$ Solvent site-site structure factor in reciprocal space, `inputfile`.svv (see http://ambermd.org/formats.html).

rout [0] Largest real space separation in Å for output files. If 0 then all grid points will be output.

kout [0] Largest reciprocal space separation in Å^{-1} for output files. If 0 then all grid points will be output.

ksave [-1] Output an intermediate solution every `ksave` steps. If `ksave <= 0` then no intermediate restart files are written. If any restart files are present at run time (`.sav` suffix) they are automatically used. However, such files are non-portable binary files.

progress [1] Write the current residue to standard output every `progress` iteration. If `progress <= 0` then residue is not reported.

selftest [0] If ‘1’, perform a self-consistency check and output the results to `inputfile.self.test`. Only tests applicable to the input parameters and system are performed. The results will depend on the input parameters (e.g., ‘tolerance’) used.

Species keywords

For each molecular species in the solvent mixture, a `species` name list should be provided.

density [] (Required.) Density of the species in M. See ’units’ below.

units [‘M’] Units for density value. Options are ‘M’ (molar), ‘mM’ (millimolar), ‘1/A^3’ (number per Å^3), ‘g/cm^3’ (g/cm^3) or ‘kg/m^3’ (kg/m^3).

model [] (Required.) Relative or absolute path to and name of the `.mdl` file with the parameters for this solvent molecule.

Solution Convergence

`rism1d` uses MDIIS to accelerate convergence. The default parameters for this method are usually near optimal but some systems can be difficult to converge. In such cases it may be useful to use a small step size (mdiis_del=0.1 or 0.2). Occasionally, the target tolerance of 10^{-12} can not be achieved. A tolerance of 10^{-10} to 10^{-11} is often sufficient but it is advisable to check how sensitive your calculations are to this.

mdiis_nvec [20] Number of MDIIS vectors to use.

mdiis_del [0.3] MDIIS step size.

mdiis_restart [10] If the current residual is `mdiis_restart` times larger than the smallest residual in memory, then the MDIIS procedure is restarted using the lowest residual solution stored in memory. Increasing this number can sometimes help convergence.

tolerance [1e-12] Target residual tolerance for the self-consistent solution.

maxstep [10000] Maximum number of iterations to converge to a solution.

extra_precision [1] Controls the use of extra precision routines at key points in the 1D-RISM solver. This can be useful for achieving low tolerances or for very large box lengths but increases computational cost. Strongly recommended for solutions with charged particles (e.g., salts).

0	No extra precision routines are used.
1	Sensitive matrix multiplication and addition routines are done in extra precision. A small computational cost is incurred.

Solvent Description

temperature [298.15] Temperature in Kelvin.

dieps [] (Required.) Dielectric constant of the solvent.

nsp [] (Required.) Number of species (molecules) in the solutions. Also indicates the number of `species` name lists to follow.

Other

smear [1.0] Charge smear parameter in Å for long range asymptotics corrections.

adbcor [0.5] Numeric parameter for DRISM.

7.4.2. Example

Mixed ionic solvent.

```
&PARAMETERS
THEORY='DRISM', CLOSURE='KH',        !Theory
    NR=16384, DR=0.025,                  !Grid size and spacing
    OUTLIST='x', ROUT=384, KOUT=0,       !Output
    MDIIS_NVEC=20, MDIIS_DEL=0.3, TOLERANCE=1.e-12,   !MDIIS
    KSAVE=-1,                            !Check pointing
    PROGRESS=1,                          !Output frequency
    MAXSTEP=10000,                       !Maximum iterations
    SMEAR=1, ADBCOR=0.5,                 !Electrostatics
    TEMPERATURE=310, DIEPS=78.497, NSP=3 !bulk solvent properties
/
&SPECIES
!SPC/E water
```

```
    DENSITY=55.296d0,                    !very close to 0.0333 1/A3
    MODEL="../../../dat/rism1d/model/SPC.mdl"
/
&SPECIES
    !Sodium
    units='mM'
    DENSITY=100,
    MODEL="../../../dat/rism1d/model/Na+.mdl"
/
&SPECIES
    !Chloride
    units='g/cm^3'
    DENSITY=35.45e-4,
    MODEL="../../../dat/rism1d/model/Cl-.mdl"
/
```

7.5. 3D-RISM in NAB

3D-RISM functionality is available in NAB and is built as part of the standard install procedure. MPI functionality for 3D-RISM in NAB requires some additional information at compile time, described in Section 7.5.5. At this time, standard molecular dynamics and minimization with non-polarizable force fields are supported.

7.5.1. Solvation Box Size

The non-periodic solvation box super-cell can be defined as variable or fixed in size. When a variable box size is used, the box size will be adjusted to maintain a minimum buffer distance between the atoms of the solute and the box boundary. This has the advantage of maintaining the smallest possible box size while adapting to changes of solute shape and orientation. Alternatively, the box size and grid spacing can be explicitly specified at run-time and used for the duration of the calculation.

Regardless of how the solvation box is defined, the "center" of the solute is placed in the middle of the box. The center of the solute and how it is placed in the solvent box is controlled with the centering keyword. Generally, centering=1 (center=center-of-mass) is the default and should be used for MD and centering=2 (center=center-of-geometry) should be used for minimization. Center-of-mass and center-of-geometry are conserved quantities in each method respectively.

Other options for solute centering are available for special situations. To restrict the absolute position of grid-points to be integer multiples of the grid-spacing (e.g., (2.5 Å,3.0 Å) for a grid spacing of 0.5 Å) use centering=3 for center-of-mass and centering=4 for center-of-geometry. To perform centering only on the first calculation (i.e., first step of MD or minimization or first frame of a trajectory analysis), use the negative integer corresponding to the desired center definition. This allows the solute to drift in the solvent box. Finally, with some care, it is possible to achieve custom centering using centering=0. Here, no solute centering is performed and the solvent grid has an origin of (0,0,0) and a center of $(\frac{x\text{-length}}{2}+dx, \frac{y\text{-length}}{2}+dy, \frac{z\text{-length}}{2}+dz)$. If you use centering=0, it is advisable to use a fixed-size solvent box.

7.5.2. I/O

All 3D-RISM options, including input and output files, are specified using `mm_options()` (see Section 39.1). Generated output files can be quite large and numerous. For each type of correlation, a separate file is produced for each solvent atom type. The frequency that files are produced is controlled by the `ntwrism` parameter. For every time step that output is produced, a new set of files is written with the time step number in the file name. For example, a molecular dynamics calculation using an SPC/E water model with `ntwrism=2` and `guvfile=guv` will produce two files on time step ten: `guv.O.10.dx` and `guv.H1.10.dx`.

7.5.3. Examples

Molecular Dynamics

```
⋮
mm_options("ntpr=100, ntpr_md=100");
mm_options("dt=0.002");                           //Large time step
mm_options("rattle=1");                           //Use RATTLE
mm_options("cut=999.0");                          //No solute-solute
                                                  //cut off
mm_options("rism=1");                             //Use 3D-RISM-KH
mm_options("xvvfile=../rism1d/spc/spc.xvv.save"); //1D-RISM input
⋮
```

Minimization

```
⋮
mm_options("ntpr=1, cut=999.0");                  //No solute-solute
                                                  //cut off
mm_options("rism=1");                             //Use 3D-RISM-KH
mm_options("xvvfile=../rism1d/spc/spc.xvv.save"); //1D-RISM input
mm_options("tolerance=1e-11");                    //Low tolerance
mm_options("solvcut=999.0");                      //No solute-solvent
                                                  //cut off
mm_options("centering=2");                        //Center solute
                                                  //using center-
                                                  //of-geometry
⋮
```

7.5.4. Thermodynamic Output

When nptrism$\neq 0$ thermodynamic data about the solvent is output as a table of solute and solvent information. When using the NAB and rism3d.snglpnt interfaces, units are indicated in the key table or as indicated below.

7.5.4.1. Solute Information

solutePotentialEnergy [kcal/mol] provides the total potential energy of the solute and its decomposition into the potential energy terms. The solvation free energy for the current 3D-RISM closure is included as this corresponds to the solvation forces the solute would experience. The energy terms, in order, are Total, LJ, Coulomb, Bond, Angle, Dihedral, H-Bond, LJ-14, Coulomb-14, Restraints, and 3D-RISM.

7.5.4.2. Solvent Information

Solvent information consists of core set of thermodynamic information and optional solvation free energy corrections. Temperature derivatives and polar/non-polar decomposition is performed when entropicDecomp and polarDecomp options are used. Temperature derivatives names have a postfix of _dT, except for free energies, which are decomposed into solvationEnergy and -TS. Polar/non-polar components have polar or apolar added to the front of the quantity name.

rism_excessChemicalPotential [kcal/mol] Excess chemical potential or solvation free energy for the selected closure (see Section 7.1.2).

rism_excessChemicalPotentialGF [kcal/mol] (Optional) Excess chemical potential or solvation free energy using the Gaussian fluctuation functional (see Eq. (7.17)).

rism_excessChemicalPotentialPCPLUS [kcal/mol] (Optional) Excess chemical potential or solvation free energy using the PC+/3D-RISM functional (see Section 7.2.4).

rism_excessChemicalPotentialUC [kcal/mol] (Optional) Excess chemical potential or solvation free energy using the UC functional (see Section 7.2.4).

rism_solventPotentialEnergy [kcal/mol] Interaction energy between the solute and solvent, calculated from

$$\Delta U_{\mathrm{sol}} = \sum_{\alpha} \rho_{\alpha} \int d\mathbf{r} g_{\alpha}^{\mathrm{UV}}(\mathbf{r}) u_{\alpha}^{\mathrm{UV}}(\mathbf{r}).$$

rism_excessParticlesCorrected [#] Excess number of solvent particles compared to a uniform distribution at bulk density.

rism_excessChargeCorrected [e] Excess charge of solvent particles compared to a uniform distribution at bulk density.

rism_KirkwoodBuff [Å^3] All space integral of the total correlation function.

rism_DCFintegral [Å^3] All space integral of the direct correlation function.

7.5.5. Compiling MPI 3D-RISM

Executables compiled with mpinab and 3D-RISM must link to both C and Fortran MPI libraries, which is not the default behavior of most MPI compilers. As there are a wide variety of MPI implementations and no standards for naming Fortran libraries, 3D-RISM is not included by default when compiling mpinab. The additional steps required to include 3D-RISM in mpinab are

1. If
   a) you are using OpenMPI 1.7 or higher or MPICH, proceed to step 2.
   b) you are *not* using OpenMPI 1.7 or higher or MPICH, identify the Fortran 77 libraries corresponding to your MPI implementation. These will be found in the lib directory for your MPI implementation and will likely contain "f" or "f77" in the file name. Set the XTRA_FLIBS environment variable to contain the compiler directive to link the library.
   For example, the OpenMPI 1.6 and MPICH2 library files are libmpi_f77.a and libfmpich.a respectively (the suffix may vary) and XTRA_FLIBS could be explicitly set as:

   OpenMPI **`export XTRA_FLIBS="lmpi_usempif08 -lmpi_mpifh`**"
   MPICH2 **`export XTRA_FLIBS=-lfmpich`**
2. Run configure and specify both -mpi and -rismmpi. For example:
   ```
   ./configure -mpi -rismmpi gnu
   ```
3. For dynamically linked executables (the default), set your LD_LIBRARY_PATH environment variable to the location of your MPI library:
   ```
   export LD_LIBRARY_PATH=$MPIHOME/lib
   ```
   where $MPIHOME is the base directory for you MPI installation.

7.6. `rism3d.snglpnt`

3D-RISM functionality is also available in the command line tools rism3d.snglpnt and rism3d.snglpnt.MPI installed at compile time. These programs perform single point 3D-RISM calculations on trajectories and individual solute snapshots. No other processing is done to the structures so unwanted solvent molecules should be removed before hand. Except for minimization and molecular dynamics, all 3D-RISM features are available. Thermodynamic data is always output (see Section 7.5.4). Note that these executables are built by NAB so please see Section 7.5.5 to ensure rism3d.snglpnt.MPI is built.

7.6.1. Usage

3D-RISM specific command line keywords generally correspond to keyword options available in NAB's mm_options (see Section 39.1). If run without input, rism3d.snglpnt prints default settings for all parameters.

`--pdb` *PDB file* (Required, input.) PDB file for the solute. Coordinates are only used if a restart or trajectory file is not supplied.

`--prmtop` *prmtop file* (Required, input.) Parameter topology file for the solute.

`--rst` *restart file* (Optional, input.) Coordinates for the solute in restart format.

`--y|traj` *trajectory file* (Optional, input.) Trajectory for the solute in NetCDF or ASCII format.

`--xvv` X^{VV} *file* (Required, input.) Bulk solvent susceptibility file from 1D-RISM (see http://ambermd.org/formats.html).

`--guv` G^{UV} *root* (Optional, output.) Root name for 3D solvent pair distribution files.

`--cuv` C^{UV} *root* (Optional, output.) Root name for 3D solvent direct correlation files.

`--huv` H^{UV} *root* (Optional, output.) Root name for 3D solvent total correlation files.

`--uuv` U^{UV} *root* (Optional, output.) Root name for 3D solvent potential [kT] files.

`--asymp` *asymptotics root* (Optional, output.) Root name for 3D real-space long range asymptotics for total and direct correlation files. This will produce one file for each of C and H for each frame requested and does not include the solvent site charge. Multiply the distribution by the solvent site charge to obtain the long-range asymptotics for that site.

`--quv` Q^{UV} *root* (Optional, output.) Root name for 3D solvent charge density distribution files. This is the charge density [$e/\text{Å}$] at each grid point with contributions from all solvent types.

`--chgdist` *charge distribution root* (Optional, output.) Root name for 3D solvent charge distribution files. This gives a point charge [e] at each grid point with contributions from all solvent types.

`--volfmt` (Optional.) Format of volumetric data files. May be dx for DX files or xyzv for XYZV format (see http://ambermd.org/formats.html).

`--closure` *closure name* (Optional.) A whitespace separated list of one or more of KH, HNC or PSE*n* where "*n*" is a positive integer. If more than one closure is provided, the 3D-RISM solver will use the closures in order to obtain a solution for the last closure in the list when no previous solutions are available. The solution for the last closure in the list is used for all output. This can be useful for difficult to converge calculations (see §7.3.1).

`--closureorder` *closure order* (Deprecated.) Specifies the order of the PSE-*n* closure if the closure name is given as "PSE" or "PSEN" (no integers).

`--noasympcorr` (Optional.) Turn off long range asymptotic corrections for thermodynamic output only. Long-range asymptotics are still used to calculate the solution.

`--buffer` *distance* (Optional.) Minimum distance between the solute and the edge of the solvent box. Use this with `--grdspc`. Incompatible with `--ng` and `--solvbox`.

`--solvcut` *distance* (Optional.) Set solute-solvent interaction cut off distance. If '-1' or no value is specified then the buffer distance is used. If a buffer distance is not provided, the cut off must be explicitly set. Note that Coulomb interactions are interpolated and not truncated beyond the cut off. See [193] for details.

`--grdspc` *3D grid spacing* (Optional.) Comma separated linear grid spacings for x, y and z dimensions. Use this with --buffer. Incompatible with `--ng` and `--solvbox`.

`--ng` *3D grid points* (Optional.) Comma separated number of grid points for *x*, *y* and *z* dimensions. Use this with --solvbox. Incompatible with `--buffer` and `--grdspc`.

`--solvbox` *3D box length* (Optional.) Comma separated solvation box side length for *x*, *y* and *z* dimensions. Use this with --ng. Incompatible with `--buffer` and `--grdspc`.

`--tolerance` *residual target* (Optional.) A whitespace separated list of maximum residual values for solution convergence. When used in combination with a list of closures it is possible to define different tolerances for each of the closures. This can be useful for difficult to converge calculations (see §7.3.1). For the sake of efficiency, it is best to use as high a tolerance as possible for all but the last closure. Three formats of list are possible.

one tolerance All closures but the last use a tolerance of 1. The last tolerance in the list is used by the last closure. In practice this, is the most efficient.

two tolerances All closures but the last use the first tolerance in the list. The last tolerance in the list is used by the last closure.

n tolerances Tolerances from the list are assigned to the closure list in order.

`--mdiis_del` *step size* (Optional.) MDIIS step size.

`--mdiis_nvec` *# of vectors* (Optional.) Number of previous iterations MDIIS uses to predict a new solution.

`--mdiis_restart` *# of vectors* (Optional.) If the current residual is `mdiis_restart` times larger than the smallest residual in memory, then the MDIIS procedure is restarted using the lowest residual solution stored in memory. Increasing this number can sometimes help convergence.

`--maxstep` *step number* (Optional.) Maximum number of iterative steps per solution.

`--npropagate` *# old solutions* (Optional.) Number of previous solutions to use in predicting a new solution.

`--polarDecomp` (Optional.) Decomposes solvation free energy into polar and non-polar components. Note that this typically requires 80% more computation time.

`--entropicDecomp` (Optional.) Decomposes solvation free energy into energy and entropy components. Also performs temperature derivatives of other calculated quantities. Note that this typically requires 80% more computation time and requires a .xvv file version 1.000 or higher (see §7.1.3 and 7.3).

`--gf` (Optional.) Compute the Gaussian fluctuation excess chemical potential functional (see §7.1.2).

`--pc+` (Optional.) Compute the PC+/3D-RISM excess chemical potential functional (see §7.2.4).

`--uccoeff` *a,b[,a1,b1]* (Optional.) Compute the UC excess chemical potential functional with the provided coefficients (see §7.2.4). *a* and *b* are the coefficients for the original UC functional, though using the closure excess chemical potential functional. *a1* and *b1* are optional and provide temperature dependence to the correction (UCT in [228]).

`--centering` *method* (Optional.) Select how solute is centered in the solvent box.

-4 Center-of-geometry with grid-point rounding. Center on first step only.

-3 Center-of-mass with grid-point rounding. Center on first step only.

-2 Center-of-geometry. Center on first step only.

-1 Center-of-mass. Center on first step only.

0 No centering. Dangerous.

1 Center-of-mass. Center on every step. Recommended for molecular dynamics.

2 Center-of-geometry. Center on every step. Recommended for minimization.

3 Center-of-mass with grid-point rounding.

4 Center-of-geometry with grid-point rounding.

`--verbose` *level* (Optional.)

0 No output.

1 Print the number of iterations required to converge.

2 Print convergence details for each iteration.

7.7. 3D-RISM in sander

3D-RISM functionality is available in *sander* and is built as part of the standard install procedure. MPI functionality for 3D-RISM in sander requires some additional information at compile time, described in Section 7.5.5. Some features specific to *sander* are discussed here.

7.7.1. Multiple Time Step Methods for 3D-RISM

At this time, the computational cost of 3D-RISM is still prohibitive for performing calculations at each step of molecular dynamics calculations. One of the most effective ways to reduce this computational burden is to reduce the number of solutions calculated by using multiple time step (MTS) methods. Two MTS methods, r-RESPA and force-coordinate extrapolation (FCE), are implemented for 3D-RISM in sander and can be combined such that solutions are only calculated once every 4 ps [235].

r-RESPA[236, 237] and I-Verlet[238] impulse MTS algorithms are widely used methods to reduce the computational load of long-range interactions while maintaining the desirable properties of energy conservation and time reversibility. Impulse MTS can be invoked for 3D-RISM independent of the existing r-RESPA implementation using the `RISMnRESPA` variable. For typical biomolecular simulations, impulse MTS is limited to a maximum step size of 8 fs if using the optimized Nose-Hoover thermostat (`ntt`=9) and 5 fs[239] for the Langevin thermostat. Since the computational load of calculating all internal interactions of the solute is small compared to the 3D-RISM calculation, it is recommend to use `dt=0.001, nrespa=1` and `RISMnRESPA`=2 or 5, depending on the integrator.

To overcome the stability limitation of impulse MTS, FCE uses one of several available extrapolation methods to efficiently predict the forces for some time steps rather than computing a full 3D-RISM solution[193, 240]. In the simplest extrapolation scheme, corresponding to `FCEntrans`=0, forces, $\{\mathbf{F}\}$, on N^{U} solute atoms for the current time step t_k are approximated as a linear combination of forces from the n previous time steps obtained from 3D-RISM calculations,

$$\{\mathbf{F}\}^{(k)} = \sum_{l=1}^{n} a_{kl}\,\{\mathbf{F}\}^{(l)}\,, \quad l \in \text{3D-RISM steps.} \tag{7.18}$$

The weight coefficients a_{kl} are obtained by expressing the current set of coordinates, $\{\mathbf{R}\}^{(k)}$, as a linear combination of coordinates from the n previous time steps for which 3D-RISM calculations were performed. That is, the current set of coordinates is projected onto the basis of n previous solute arrangements by minimizing the norm of the difference between the current $3 \times N^{\mathrm{U}}$ matrix of coordinates $\{\mathbf{R}\}^{(k)}$ and the corresponding linear combination of the previous ones $\{\mathbf{R}\}^{(l)}$,

$$\text{minimize} \left| \{\mathbf{R}\}^{(k)} - \sum_{l=1}^{n} a_{kl}\,\{\mathbf{R}\}^{(l)} \right|^2 .$$

Coefficients a_{kl} are then used in Equation (7.18)to extrapolate forces at the current intermediate time step. Similarly, the known coordinates for the current time step can be approximated from previous time steps as

$$\{\mathbf{R}\}^{(k)} = \sum_{l=1}^{N} a_{kl}\,\{\mathbf{R}\}^{(l)}\,.$$

Five extrapolation methods are available (`FCEntrans`=0-4, see below) and each differs in computational cost along with the largest permitted outer time step, ranging from 20 fs (`FCEntrans`=4 with Langevin dynamics,

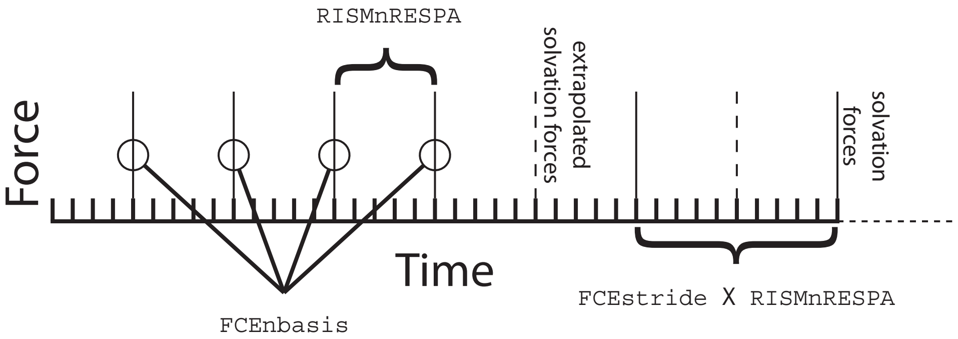


Figure 7.1.: *Multiple time step methods in 3D-RISM.* `RISMnRESPA`$(= 5)$ *is the number of base time steps between application of solvation forces (exact or extrapolated).* `FCEnbasis`$(= 4)$ *is the number of previous solutions used to extrapolate forces, in this case four previous solutions. Once* `FCEnbasis` *solutions have be calculated, exact 3D-RISM forces are calculated every* `FCEstride`$(= 2)\times$`RISMnRESPA` *time steps; solvation forces are otherwise obtained through extrapolation.*

`ntt`=3) all the way up to 4 ps (`FCEntrans`=6 using OIN, `ntt`=9). The latter procedures utilize a more complex extrapolation protocol than pictured above, involving a rotation of the outer basis coordinates and coefficient weight normalization and minimization. For a detailed description of these methods, please refer to [240] and [235]. Note that FCE MTS does not conserve energy and is not time reversible.

Combined impulse FCE MTS calculations (see Figure 7.1) start the simulation using impulse MTS, where full RISM-3D solutions are computed every `RISMnRESPA` time steps until the requested size for the basis set, `FCEnbasis`, is achieved. After a large enough basis set is collected, 3D-RISM calculations are only performed once every `FCEstride` × `RISMnRESPA` time steps, and `FCEnbase` of `FCEnbasis` saved coordinates are used for one of the above extrapolation procedures every `RISMnRESPA` intermediate time steps. The `FCEnbase` coordinates represent an optimized subset of `FCEnbasis`, found through distance minimization with the current solute coordinate. Note that large inaccuracies in the force extrapolation can ensue if `FCEnbase` is equal to the number of solute degrees freedom.

7.7.2. 3D-RISM in `sander`

Full 3D-RISM functionality is available in sander as part of the standard install procedure. However, some methods available in `sander` are not compatible with 3D-RISM, such as QM/MM simulations. At this time, only standard molecular dynamics, minimization and trajectory post-processing with non-polarizable force fields are supported. With the exception of multiple time step features, 3D-RISM keywords in sander are identical to those in NAB, rism3d.snglpnt and MMPBSA.py.

3D-RISM specific command line options for sander are

```
sander [standard options] -xvv xvvfile -guv guvroot -huv huvroot

    -cuv cuvroot -uuv uuvroot -asymp asympfile
    -quv quvroot -chgdist chgdistroot
```

xvvfile *input* description of bulk solvent properties, required for 3D-RISM calculations. Produced by `rism1d`.

guvroot *output* rootname for solute-solvent 3D pair distribution function, $G^{\mathrm{UV}}(\mathbf{R})$. This will produce one file for each solvent atom type for each frame requested.

huvroot *output* rootname for solute-solvent 3D total correlation function, $H^{\mathrm{UV}}(\mathbf{R})$. This will produce one file for each solvent atom type for each frame requested.

cuvroot *output* rootname for solute-solvent 3D total correlation function, $C^{\mathrm{UV}}(\mathbf{R})$. This will produce one file for each solvent atom type for each frame requested.

uuvroot *output* rootname for solute-solvent 3D potential energy function, $U^{\mathrm{UV}}(\mathbf{R})$, in units of kT. This will produce one file for each solvent atom type for each frame requested.

asympfile *output* rootname for solute-solvent 3D long-range real-space asymptotics for C and H. This will produce one file for each of C and H for each frame requested and does not include the solvent site charge. Multiply the distribution by the solvent site charge to obtain the long-range asymptotics for that site.

quvroot *output* rootname for solute-solvent 3D charge density distribution [e/Å]. This will produce one file that combines contributions from all solvent atom types for each frame requested.

chgdist *output* rootname for solute-solvent 3D charge distribution [e]. This will produce one file that combines contributions from all solvent atom types for each frame requested.

Generated output files can be large and numerous. For each type of correlation, a separate file is produced for each solvent atom type. The frequency that files are produced is controlled by the `ntwrism` parameter. Every time step that output is produced, a new set of files is written with the time step number in the file name. For example, a molecular dynamics calculation using an SPC/E water model with `ntwrism=2` and `-guv guv` on the command line will produce two files on time step ten: `guv.O.10.dx` and `guv.H1.10.dx`.

7.7.2.1. Keywords

With the exception of irism, which is found in the &cntrl name list, all 3D-RISM options are specified in the &rism name list.

irism [0] Use 3D-RISM. Found in `&cntrl` name list.

= 0 Off.

= 1 On.

Closure Approximation

closure [KH] Comma separate list of closure approximations. If more than one closure is provided, the 3D-RISM solver will use the closures in order to obtain a solution for the last closure in the list when no previous solutions are available. The solution for the last closure in the list is used for all output.

= KH Kovalenko-Hirata (KH).

= HNC Hyper-netted chain equation (HNC).

=PSE*n* Partial series expansion of order-*n* (PSE-*n*), where “*n*” is a positive integer.

Solvation Free Energy Corrections

gfCorrection [0] Compute the Gaussian fluctuation excess chemical potential functional (see §7.1.2).

= 0 Off.

= 1 On.

pcpluscorrection [0] Compute the PC+/3D-RISM excess chemical potential functional (see §7.2.4).

= 0 Off.

= 1 On.

uccoeff [0,0,0,0] Compute the UC excess chemical potential functional with the provided coefficients (see §7.2.4). *a* and *b* are the coefficients for the original UC functional, though using the closure excess chemical potential functional. *a1* and *b1* are optional and provide temperature dependence to the correction (UCT in [228]).

Long-range asymptotics Long-range asymptotics are used to analytically account for solvent distribution beyond the solvent box. Long-range asymptotics are always used to when calculating solution but can be omitted for the subsequent thermodynamic calculations, though it is not recommended.

asympcorr [T] Use long-range asymptotic corrections for thermodynamic calculations.

= T Use the long-range corrections.

= F Do not use long-range corrections.

Solvation Box The non-periodic solvation box super-cell can be defined as variable or fixed in size. When a variable box size is used, the box size will be adjusted to maintain a minimum buffer distance between the atoms of the solute and the box boundary. This has the advantage of maintaining the smallest possible box size while adapting to changes of solute shape and orientation. Alternatively, the box size can be specified at run-time. This box size will be used for the duration of the sander calculation.

solvcut [`buffer`] Cut-off distance for solvent-solute potential and force calculations. If `buffer` < 0 and `solvcut` is not explicitly set, `solvcut` $= |$`buffer`$|$. For minimization it is recommended to not use a cut-off (e.g. `solvcut=9999`).

Variable Box Size

buffer [14] Minimum distance in Å between the solute and the edge of the solvent box.

< 0 Use fixed box size (`ng3` and `solvbox`).

>= 0 Buffer distance.

grdspc [0.5,0.5,0.5] Linear grid spacing in Å.

Fixed Box Size

ng3 [] Sets the number of grid points for a fixed size solvation box. This is only used if buffer< 0.

nx,ny,nz Points for x, y and z dimensions.

solvbox [] Sets the size in Å of the fixed size solvation box. This is only used if buffer< 0.

lx,ly,lz Box length in x, y and z dimensions.

Solution Convergence

tolerance [1e-5] A list of maximum residual values for solution convergence. When used in combination with a list of closures it is possible to define different tolerances for each of the closures. This can be useful for difficult to converge calculations (see Subsection 7.4.1 for details). For the sake of efficiency, it is best to use as high a tolerance as possible for all but the last closure. For minimization a tolerance of 1e-11 or lower is recommended. Three formats of list are possible.

one tolerance All closures but the last use a tolerance of 1. The last tolerance in the list is used by the last closure. In practice this, is the most efficient.

two tolerances All closures but the last use the first tolerance in the list. The last tolerance in the list is used by the last closure.

n tolerances Tolerances from the list are assigned to the closure list in order.

mdiis_del [0.7] "Step size" in MDIIS.

mdiis_nvec [5] Number of vectors used by the MDIIS method. Higher values for this parameter can greatly increase memory requirements but may also accelerate convergence.

mdiis_restart [10] If the current residual is `mdiis_restart` times larger than the smallest residual in memory, then the MDIIS procedure is restarted using the lowest residual solution stored in memory. Increasing this number can sometimes help convergence.

mdiis_method [2] Specify implementation of the MDIIS routine.

= 0 Original. For small systems (e.g. $< 64^3$ grid points) this implementation may be faster than the BLAS optimized version.

= 1 BLAS optimized.

= 2 BLAS and memory optimized.

maxstep [10000] Maximum number of iterations allowed to converge on a solution.`nrespa`

npropagate [5] Number of previous solutions propagated forward to create an initial guess for this solute atom configuration.

= 0 Do not use any previous solutions

= 1..5 Values greater than 0 but less than 4 or 5 will use less system memory but may introduce artifacts to the solution (e.g., energy drift).

Minimization and Molecular Dynamics

centering [1] Controls how the solute is centered/re-centered in the solvent box.

= -4 Center-of-geometry with grid-point rounding. Center on first step only.

= -3 Center-of-mass with grid-point rounding. Center on first step only.

= -2 Center-of-geometry. Center on first step only.

= -1 Center-of-mass. Center on first step only.

= 0 No centering. Dangerous.

= 1 Center-of-mass. Center on every step. Recommended for molecular dynamics.

= 2 Center-of-geometry. Center on every step. Recommended for minimization.

= 3 Center-of-mass with grid-point rounding.

= 4 Center-of-geometry with grid-point rounding.

zerofrc [1] Redistribute solvent forces across the solute such that the net solvation force on the solute is zero.

= 0 Unmodified forces.

= 1 Zero net force.

Trajectory Post-Processing

apply_rism_force [1] Calculate and use solvation forces from 3D-RISM. Not calculating these forces can save computation time and is useful for trajectory post-processing.

= 0 Do not calculate forces.

= 1 Calculate forces.

Multiple Time Steps Multiple time step features are only available in `sander`.

rismnrespa [1] `rismnrespa` × `dt` =RISM RESPA multiple time step. 8 fs is the maximum time step if using optimized-isokinetic integrator (ntt=9), and 5 fs using Langevin dynamics (ntt=3). "1" corresponds to no multiple time stepping.

fcestride [0] `fcestride` × `rismnrespa` × `dt` = FCE multiple time step, also called outer time step, i.e., full 3D-RISM solutions are performed every `fcestride` × `rismnrespa` steps. In between full solutions extrapolated force impulses are applied every `rismnrespa` steps. "1" corresponds to no multiple time stepping.

= 0 No FCE multiple time stepping.

= 1 Invokes the FCE code but yields the same trajectories as 0.

>= 1 Invoke FCE with 3D-RISM solutions every `fcestride` × `rismnrespa` steps.

fcenbasis [20] Number of previous full solutions to store, `fcenbase` of these are used for the force extrapolation. If FCE is not desired this can be set to 1 to reduce memory usage.

fcenbase [20] The number of previous solutions to use for the force extrapolation. This is a subset of `fcenbasis` and must be <= `fcenbasis`. If `fcenbase` < `fcenbasis`, then an optimized subset of fcenbasis is found through minimization of the square distances with the current coordinate - the `fcenbase` closest solutions are chosen. Options for this selection can be found in the commands that follow.

fcesort [0] Sort the `fcenbase` basis vectors for the extrapolation according to increasing distance from the current coordinate. May decrease roundoff errors.

= 0 No sorting is performed (default).

= 1 Sorting is performed.

fcecrd [0] The coordinates used for the FCE method.

= 0 The absolute x, y, z position of each neighbor atom (with translations due to `centering`).

= 1 For predicting the forces on atom *i*, use the distance of each neighbor atom as the "coordinate". This has one third the number of coordinates to use in the prediction. Also, directional information is lost.

= 2 For predicting the forces on atom *i*, use the x, y, z position of each neighbor atom with atom *i* as the origin. Recommended.

fceweigh [0] Use weighted coordinates for the force extrapolation. Works with fcetrans = [1], [2], or [3].

= 0 No weighting of the coordinates is performed (default).

= 1 Weighting of basis coordinates in the extrapolation. Expensive but more precise.

fceenormsw [0] Balancing minimization of the squared norm of the basis expansion coefficients from least squares fitting. Specifies the magnitude of the parameter ε^2 of an additional constraint added to the least squares fitting problem that balances the equations and resulting coefficients, improving the quality and stability of the force extrapolation. Used only if `fcetrans`=2.

= 0 No weight minimization is performed (default).

> 0 Minimization is performed with specified balancing parameter `fceenormsw.` This parameter should in general be small as the squared norm is being minimized, and should be optimized to the value that produces the most accurate results from simulation.

fcetrans [0] The method of transformation of the outer basis coordinates and the method of finding expansion coefficients in the least squares minimization problem. It can significantly affect the permitted size of the outer time step. Transformations involve a non-Eckhart rotation of all `fcenbasis` coordinates. In the least squares minimization problem, for the QR decomposition method, normalization is used if `fcenbase` > solute degrees of freedom.

= 0 (Default) No coordinate transformation of the outer basis coordinates. Fast but not precise and should only be invoked if using small outer time steps (up to 200fs). Method of QR decomposition is used for finding expansion coefficients from least squares minimization.

= 1 Transformation of basis coordinates with respect to the first (most recent) basis coordinate, from these the `fcenbase` subset is selected by minimum distance from current (also rotated) coordinate. QR decomposition is used for the least squares minimization. Permits large outer time steps on the order of several picoseconds. Fastest with regard to [2] and [3].

= 2 ASFE extrapolation: like [1], transformation of basis coordinates with respect to first basis point, but normal equations method is utilized instead of QR, with additional squared norm minimization, specified by `fceenormsw`. An extra precision and stability is gained with small, positive values of `fcernormsw`. Most advanced method in Amber 15. This represents the ASFE extrapolation scheme as laid out in [240].

= 3 (place holder, same as 2 above)

= 4 Basic force extrapolation - no coordinate transformation, weighting, selecting, and sorting. Only small outer time steps, on the order of tens of fs, are permitted. This is the method as implemented in Amber 11.

= 5 GSFE extrapolation 1: Individual transformation and selecting with respect to the current coordinate of each atom using a neighbouring scheme complemented by the e-minimization and *ifreq*-scheme (see *fceifreq* below) as well as all other developed techniques. It is recommended for large macromolecules of greater than 10 Å in size and can be used with very large outer steps (up to order of several picoseconds). See [241]for detailed explaination. This represents the one of the two new GSFE extrapolation schemes (Generalized Solvent Force-coordinate Extrapolation) as presented in [241].

= 6 GSFE extrapolation 2: Individual transformation and selecting with respect to the post coordinate of each atom using a neighbouring scheme compemented by the e-minimization and the full *ifreq*-support. It is recommended for large macromolecules and can be used with huge outer steps (up to order of several picoseconds). It appears to be better than the above case *fcetrans=5* (partial *ifreq*-support version) because it can be exploited with larger number (up to N~100-200) of basic points providing a higher accuracy (with nearly the same computational efforts as the *fcetrans*=5-version at N~30), but may require more memory. Note that at any values of *fceifreq*, both the approaches have the same scheme for building the index mask which maps the extended set to the best subset and differ in the way of constructing the transformation matrix. At *fceifreq*=1, these two approaches are equivalent. This is the second GSFE scheme presented in [241] and [235] and represents the most advanced 3D-RISM solvent force extrapolation scheme available in AMBER to date.

fceifreq Extended to basic maping list updating frequency used in the GSFE FCE extrapolation schemes above. If fceifreq=1 then fcetrans=6 is equivalent to fcetrans=5. See [241]for detailed explaination. Default value is 1.

fcentfrcor Net force correction flag for GSFE force extrapolation (fcetrans=5 and fcetrans=6). If fcentfrcor > 0, a correction factor is subtracted from the extrapolated forces. See [241]for in depth explaination. Default is 0.

Output

ntwrism [0] Indicates that solvent density grid should be written to file every `ntwrism` iterations.

= 0 No files written.

>= 1 Output every `ntwrism` time steps.

volfmt [‘DX’] Format of volumetric data files. May be ’dx’ for DX files or ’xyzv’ for XYZV format. See the AmberTools manual for more information.

verbose [0] Indicates level of diagnostic detail about the calculation written to the log file.

= 0 No output.

= 1 Print the number of iterations used to converge.

= 2 Print details for each iteration and information about what FCE is doing every `progress` iterations.

write_thermo [1] Print solvation thermodynamics in addition to standard sander output. The format is the same as that found in NAB and rism3d.snglpnt.

polarDecomp [0] Decomposes solvation free energy into polar and non-polar components. Note that this typically requires 80% more computation time.

= 0 No polar/non-polar decomposition.

= 1 Polar/non-polar decomposition.

entropicDecomp [0] Decomposes solvation free energy into energy and entropy components. Also performs temperature derivatives of other calculated quantities. Note that this typically requires 80% more computation time and requires a .xvv file version 1.000 or higher (see §7.1.3 and 7.3).

= 0 No entropic decomposition.

= 1 Entropic decomposition.

progress [1] Display progress of the 3D-RISM solution every `kshow` iterations. 0 indicates this information will not be displayed. Must be used with `verbose` > 1.

7.7.2.2. `Example`

Molecular Dynamics (`imin=0`)

```
molecular dynamics with 3D-RISM and impulse MTS
&cntrl
    ntx=1, ntpr=100, ntwx=1000,ntwr=10000,
    nstlim=10000,dt=0.001,              !No shake or r-RESPA
    ntt=3, temp0=300, gamma_ln=20,      !Langevin dynamics
    ntb=0,                              !Non-periodic
    cut=999.,                           !Calculate all
                                        !solute-solute
                                        !interactions
    irism=1,
/
&rism
    rismnrespa=5,                       !r-RESPA MTS
    fcenbasis=10,fcestride=2,fcecrd=2   !FCE MTS
/
```

Minimization (`imin=1`)

```
Default XMIN minimization with 3D-RISM
&cntrl
    imin=1, maxcyc=200,
    drms=1e-3,          !RMS force. Can be as low as 1e-4
    ntmin=3,            !XMIN
    ntpr=5,
    ntb=0,              !Non-periodic
    cut=999.,            !Calculate all
                         !solute-solute interactions
    irism=1
/
&rism
    tolerance=1e-11,     !Low tolerance
    solvcut=9999,        !No cut-off for
                         !solute-solvent interactions
    centering=2          !Solvation box centering
                         !using center-of-geometry
/
```

Trajectory Post-Processing (`imin=5`)

```
Trajectory post-processing with 3D-RISM
&cntrl
    ntx=1, ntpr=1, ntwx=1,
    imin=5,maxcyc=1,        !Single-point energy calculation
                            !on each frame
    ntb=0,                  !Non-periodic
    cut=9999.,              !Calculate all
                            !solute-solute interactions
    irism=1
/
&rism
    tolerance=1e-4,         !Saves some time compared to 1e-5
    apply_rism_force=0,     !Saves some time. Forces are not used.
    npropagate=1            !Saves some time and 4*8*Nbox bytes
                            !of memory compared to npropagate=5.
/
```

7.8. RISM File Formats

7.8.1. MDL

Solvent MoDeL (MDL) files use the prmtop specification. Each of the following sections may appear in the file in any order. The Fortran string format specifications can be different from the recommend values below.

```
%VERSION VERSION_STAMP = Vxxxx.yyy DATE = mm:dd:yy hh:mm:ss
```

The current version of the format is 0001.000. Date should be the date and time the file is created.

```
%FLAG TITLE
%FORMAT(20a4)
```

Optional description of the file.

```
%FLAG POINTERS
%FORMAT(10I8)
```

Defines the lengths of arrays in the file.

NATOM	Number of physical atoms in the model.
NSITE	Number of unique solvent sites (share common Lennard-Jones parameters and partial charges).

```
%FLAG ATMNAME
%FORMAT(20a4)
```

CHARACTER(len=4)(NSITE) Four character name of each solvent site.

```
%FLAG MASS
%FORMAT(5e16.8)
```

REAL*8(NSITE) Mass of each solvent site (amu).

```
%FLAG CHG
%FORMAT(5e16.8)
```

REAL*8(NSITE) Partial charge for each solvent site, $18.2223e$ ($\sqrt{kT\text{Å}}$).

```
%FLAG LJEPSILON
%FORMAT(5e16.8)
```

REAL*8(NSITE) Lennard-Jones ε for each solvent site (kcal/mol).

```
%FLAG LJSIGMA
%FORMAT(5e16.8)
```

REAL*8(NSITE) Lennard-Jones $r_{\min}/2$ (sometimes called $\sigma^*/2$) for each solvent site (Å)

$$U^{\mathrm{LJ}}_{\alpha\gamma} = \sqrt{\varepsilon_\alpha \varepsilon_\gamma}\left(\left(\frac{r_{\min,\alpha}+r_{\min,\gamma}}{2r}\right)^{12} - 2\left(\frac{r_{\min,\alpha}+r_{\min,\gamma}}{2r}\right)^{6}\right).$$

Note that this is related to the commonly used σ as

$$\sigma = r_{\min}2^{-1/6}.$$

```
%FLAG MULTI
%FORMAT(10I8)
```

INTEGER*4(NSITE) Multiplicity of each solvent site. This should sum to NATOM.

```
%FLAG COORD
%FORMAT(5e16.8)
```

REAL*8(3*NATOM) *xyz*-coordinates of each atom (Å).

7.8.2. XVV

The .xvv file provides all of the bulk-solvent information required for 3D-RISM. This includes information about the solvent model, thermodynamic state and the necessary correlation functions. .xvv files use the prmtop specification. Each of the following sections may appear in the file in any order. The format specifications can be different from the recommend values below.

1D- and 3D-RISM now use version 1.000 of the file format. Differences include

- additional information about solvent, such as mass, number of sites per species, coordinates;
- RISM's internal system of units is now used;
- temperature derivative, `DELHV0_DT` and `XVV_DT`, are included when available (see 7.4.1);
- and `SIGV` has been replaced by `RMIN2V`.

All 3D-RISM interfaces still support the original 0.001 version of the format. For detailed information on version 0.001, please see the AmberTools 1.5 manual.

```
%VERSION VERSION_STAMP = V0001.000 DATE = mm:dd:yy hh:mm:ss
```

The current version of the format is 0001.000. Date should be the date and time the file is created.

```
%FLAG POINTERS
%FORMAT(10I8)
```

Defines the lengths of arrays in the file.

NR	Number of 1D grid points in $\chi_{ab}^{\mathrm{VV}}(k)$.
NV	Number of total solvent sites.
NSP	Number of solvent species (molecules).

```
%FLAG THERMO
%FORMAT(1PE24.16)
```

REAL(8)(6) Temperature [K], dielectric constant, inverse Debye length (κ) [Å], compressibility [$\mathrm{Å}^{-3}$], grid spacing [Å], charge smear [Å].

```
%FLAG ATOM_NAME
%FORMAT(20A4)
```

CHARACTER(len=4)(NSITE) Four character name of each solvent site.

```
%FLAG MTV
%FORMAT(10I8)
```

INTEGER(4)(NSITE) Multiplicity of each solvent site.

```
%FLAG NVSP
%FORMAT(10I8)
```

INTEGER(4)(NSP) Number of sites for each solvent species.

```
%FLAG MASS
%FORMAT(1P5E16.8)
```

REAL(8)(NSITE) Mass of each solvent site (g/mol).

```
%FLAG RHOV
%FORMAT(1P5E16.8)
```

REAL(8)(NSITE) Number density of each solvent site ($\mathrm{Å}^{-3}$).

```
%FLAG QV
%FORMAT(1P5E16.8)
```

REAL(8)(NSITE) Partial charge for each solvent site multiplied by the square root of the Coulomb constant, ~18.2223 ($\sqrt{kT\mathrm{Å}}$).

```
%FLAG QSPV
%FORMAT(1P5E16.8)
```

REAL(8)(NSPECIES) Net charge for each solvent species multiplied by the square root of the Coulomb constant, ~18.2223 ($\sqrt{kT\text{Å}}$).

```
%FLAG EPSV
%FORMAT(1P5E16.8)
```

REAL(8)(NSITE) Lennard-Jones ε for each solvent site (kT).

```
%FLAG RMIN2V
%FORMAT(1P5E16.8)
```

REAL(8)(NSITE) Lennard-Jones $r_{\min}/2$ ($\sigma^{*}/2$) for each solvent site (Å).

```
%FLAG DELHV0
%FORMAT(1P5E16.8)
```

REAL(8)(NSITE) Long range Coulomb correction for each solvent site ($\sqrt{kT\text{Å}}$).

```
%FLAG DELHV0_DT
%FORMAT(1P5E16.8)
```

REAL(8)(NSITE) (Optional) Temperature derivative long range Coulomb correction for each solvent site ($\sqrt{kT\text{Å}}$).

```
%FLAG COORD
%FORMAT(1P5E16.8)
```

REAL(8)(3*sum(MTV)) Coordinates of all atoms (not sites) for each solvent species with the dipole moment aligned with the z-axis (Å).

```
%FLAG XVV
%FORMAT(1P5E16.8)
```

REAL(8)(NR,NSITE,NSITE) $\chi_{ab}^{\mathrm{VV}}(k)$. This array is stored in column major order. That is, the NR index varies fastest.

```
%FLAG XVV_DT
%FORMAT(1P5E16.8)
```

REAL(8)(NR,NSITE,NSITE) (Optional) $\delta_T \chi_{ab}^{\mathrm{VV}}(k)$. This array is stored in column major order. That is, the NR index varies fastest.

7.8.3. Site-site functionals

All .*vv files, except .xvv (see §7.8.2), provide the separation dependence of all site-site pairings for a particular functional and use the same format. The first four lines have a “#” in the first character column, provide a description of the contents of the file and indicate site-site pairs. The first data column is the site-site separation and the remaining columns provide the value of the functional for the site-site pair at this separation.

The following example is for the direct correlation function (.cvv) for pure water. A standard, ‘two-site’ water model is used, consisting of oxygen (O) and hydrogen (H1). This gives one solvent species with two atoms.

```
#RISM1D ATOM-ATOM INTERACTIONS: DIRECT CORRELATION VS. SEPARATION [A]
#S=SPECIES, A=ATOM
#    SEPARATION       S1A1:S1A1        S1A1:S1A2        S1A2:S1A2
#    SEPARATION         O:O              H1:O              H1:H1
   0.00000000E+000 -3.81875841E+002  1.64156197E+002 -9.24562553E+001
   2.50000000E-002 -3.81695327E+002  1.64139031E+002 -9.24384608E+001
```

7.8.4. Thermodynamics

Thermodynamic output is divided into global, species and site properties sections. Global properties are generally not decomposable into species or site contributions (e.g., pressure). Species properties are the values for individual molecular species, for example, the excess chemical potential of a single molecule. Some of these properties, such as the partial molar volume, may not be decomposible into individual sites. Site properties are contributions from individual sites. Values for sites from the same species will sum to give the total value for the species.

The file format is white-space delimited with the first three columns giving a description, variable name and units of the property calculated. The remaining columns contain the calculated values for the system, species or site. Descriptive lines are indicated with a leading "#".

The following example is for a standard, 'two-site' water model is used, consisting of oxygen (O) and hydrogen (H1), at standard temperature and density. In this calculation, energy/entropy free energy decomposition is also performed. I.e., `EXCHEMsp` $=$ `ESOLVsp` $-$ `TSsp`.

```
#Global properties
#Description                     Variable  Units        Value
Compressibility                  xi        [10e-4/MPa]  4.73552130E+000
Pressure_(Virial)                Pvir      [MPa]        2.51627507E+003
Excess_free_energy               FE        [kcal/mol]  -1.03698038E+003
#Species properties
#Description                     Variable  Units        SPC
Excess_chemical_potential        EXCHEMsp  [kcal/mol]  -2.79190339E+000
Solvation_energy                 ESOLVsp   [kcal/mol]  -1.16421825E+001
-Temperature*solvation_entropy   -TSsp     [kcal/mol]   8.85027911E+000
Partial_molar_volume             PMV       [A^-3]       3.00300236E+001
#Site properties
#Description                     Variable  Units       O                H1
Excess_chemical_potential        EXCHEMv   [kcal/mol]  -6.47897321E+000  3.68706981E+000
Solvation_energy                 ESOLVv    [kcal/mol]  -1.19565867E+001  3.14404192E-001
-Temperature*solvation_entropy   -TSv      [kcal/mol]   5.47761350E+000  3.37266562E+000
```

7.8.5. Total excess values

.n00 and .q00 files provide the total excess coordination number and charge about each solvent site. The total excess of site γ around site α is

$$n_{\alpha\gamma}^{\text{extot}} = \rho_\gamma \int_0^\infty h_{\alpha\gamma}(r)\,dr,$$

while the total excess charge is

$$q_{\alpha\gamma}^{\text{extot}} = q_\gamma n_{\alpha\gamma}^{\text{extot}}.$$

These values are presented in their respective files as $n_{\text{site}} \times n_{\text{site}}$ arrays. Any asymmetry in these arrays is due to numerical error. .q00 files additionally provided the total excess charge from all sites.

The following example gives the total excess charge for a standard, 'two-site' water model is used, consisting of oxygen (O) and hydrogen (H1), at standard temperature and density.

```
#Total excess coordinated charge [e] of column site about row site
            O                H1           Total charge
O     7.92607232E-001 -7.92607230E-001  1.67181313E-009
H1    7.92607231E-001 -7.92607229E-001  2.44386922E-009
```

7.8.6. DX volumetric data

By default, 3D correlation functions from 3D-RISM calculations use the ASCII version of the Data Explorer (DX) file format for volumetric data on regular grids as defined in the DX user manual: `http://opendx.informatics.jax.org/docs/html/pages/usrgu068.htm#HDREDF`.

Header

```
object 1 class gridpositions counts Nx Ny Nz
```

Nx	INTEGER*4. Number of grid points in the x dimension.
Ny	INTEGER*4. Number of grid points in the y dimension.
Nz	INTEGER*4. Number of grid points in the z dimension.

```
origin Ox Oy Oz
```

Ox	REAL*8. x coordinate of grid origin in Cartesian space.
Oy	REAL*8. y coordinate of grid origin in Cartesian space.
Oz	REAL*8. z coordinate of grid origin in Cartesian space.

```
delta dx 0 0
delta 0 dy 0
delta 0 0 dz
```

dx	REAL*8. Linear grid size between in the x dimension.
dy	REAL*8. Linear grid size between in the y dimension.
dz	REAL*8. Linear grid size between in the z dimension.

```
object 2 class gridconnections counts Nx Ny Nz
object 3 class array type double rank 0 items N data follows
```

N	INTEGER*4. $\mathtt{N} = \mathtt{Nx} \times \mathtt{Ny} \times \mathtt{Nz}$.

Data

```
data(i,j,k) data(i,j,k+1) data(i,j,k+2)
```

data(i,j,k)	REAL*8. Three data values per line with the last (z) index varying fastest for a total of N values.

Footer

```
object "Untitled" call field
```

7.8.7. XYZV volumetric data

An alternate format for volumetric data is the simple ASCII x-y-z-value (XYZV) format. The x-, y- and z-coordinates each grid point is written on a line followed by the value of the grid point. There is no header or footer. For example,

```
⋮
-7.10789855E+000 -1.12570084E+001 -1.61284113E+001  1.35771922E-006
-2.10789855E+000 -1.12570084E+001 -1.61284113E+001 -5.32279347E-006
 2.89210145E+000 -1.12570084E+001 -1.61284113E+001 -1.58802759E-005
```

8. Empirical Valence Bond

8.1. Introduction

Chemical reactivity can be formulated within the empirical valence bond (EVB) model[242, 243], whereby the reactive surface is defined as the lowest adiabatic surface obtained by diagonalization of the potential energy matrix in the representation of non-reactive diabatic states. These diabatic states can be described by a force field approach, such as Amber, or by a prescription incorporating information from ab initio calculations. The coupling elements in the matrix embody all the physics needed for describing transitions between the diabatic states.

As an example, the intramolecular proton transfer reaction in malonaldehyde (Figure 8.1) can be described by a two-state EVB matrix

$$\mathbf{V} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \tag{8.1}$$

where valence bond state 1 represents the reactant state (RS) with the proton `H9` bonded to `O8` and valence bond state 2 represents the product state (PS) with the proton bonded to `O7`. The matrix elements V_{11} and V_{22} are simply the energies of the reactant and product systems. The off-diagonal elements of this symmetric matrix, i.e. $V_{12} = V_{21}$, couple these diabatic states.

Amber provides several options for computing the V_{12} resonance integrals. In its simplest form, V_{12} is set to a constant value which provides an EVB surface that reproduces experimental or ab initio barrier heights. More flexibility can be introduced into V_{12} by employing an exponential or Gaussian function of the coordinates. It has recently been shown [244, 245] that a linear combination of distributed Gaussian functions is the most accurate and flexible form for V_{12}. With a set of distributed Gaussians, V_{12} can be fit to high-level electronic structure data using the following form,

$$V_{12}^2(\mathbf{q}) = \sum_K \sum_{i \geq j \geq 0}^{NDim} B_{ijK}\, g\left(\mathbf{q}, \mathbf{q}_K, i, j, \alpha_K\right) \tag{8.2}$$

$$V_{12}^2(\mathbf{q}) = \left[V_{11}(\mathbf{q}) - V(\mathbf{q})\right]\left[V_{22}(\mathbf{q}) - V(\mathbf{q})\right] \tag{8.3}$$

$$g\left(\mathbf{q}, \mathbf{q}_K, 0, 0, \alpha_K\right) = \left(1 + \frac{1}{2}\alpha_K |\mathbf{q} - \mathbf{q}_K|^2\right) \exp\left[-\frac{1}{2}\alpha_K |\mathbf{q} - \mathbf{q}_K|^2\right] \tag{8.4}$$

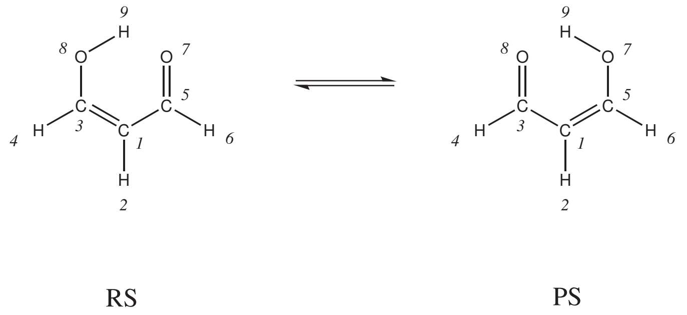


Figure 8.1.: *Intramolecular proton transfer in malonaldehyde.*

$$g(\mathbf{q},\mathbf{q}_K,i,0,\alpha_K) = (\mathbf{q}-\mathbf{q}_K)_i \exp\left[-\frac{1}{2}\alpha_K|\mathbf{q}-\mathbf{q}_K|^2\right] \tag{8.5}$$

$$g(\mathbf{q},\mathbf{q}_K,i,j,\alpha_K) = (\mathbf{q}-\mathbf{q}_K)_i (\mathbf{q}-\mathbf{q}_K)_j \exp\left[-\frac{1}{2}\alpha_K|\mathbf{q}-\mathbf{q}_K|^2\right] \tag{8.6}$$

where $g(\mathbf{q},\mathbf{q}_K,i,j,\alpha_K)$ are s-, p-, and d-type Gaussians at a number of points, $\mathbf{q}_K$, on the potential energy surface, *NDim* is the total number of internal coordinates, V is the ab initio energy and **B** is a vector of coefficients. It is important to note that a nonstandard s-type Gaussian is employed to precondition the resulting set of linear equations that is passed to a GMRES[246] (aka DIIS[247, 248]) solver. For a more exhaustive discussion of the DG EVB method please see reference [245]. Additionally, the EVB facility in Amber can perform MD or energy optimization on the EVB ground-state surface and biased sampling along a predefined reaction coordinate (RC). Nuclear quantization based on the Feynman path integral formalism [249–251] is also possible.

8.2. General usage description

The EVB facility is built on top of the *multisander* infrastructure in Amber. (Section 17.11) As such, the user will need to build the parallel version of *sander* in order to utilize the EVB feature. Information for each EVB diabatic state is obtained from separate (simultaneous) instances of *sander*. The energies and forces of all the states are communicated via MPI to the master node, which is responsible for computing the EVB energy and forces and broadcasting these to the other nodes for the next MD step.

The required input files are (1) an EVB *multisander* group file containing per line all the command line options for each *sander* job, (2) the *mdin*, coordinate, and parmtop files specified in the group file, and (3) the EVB input files. At the top level, an EVB calculation is invoked as follows:

```
mpirun -np <# procs> sander.MPI -ng <# groups> -groupfile <EVB group file>
```

The contents of the EVB group file is similar to that for a conventional *multisander* execution, with the addition of a command line flag *-evbin* for specifying the name of the EVB input file. Below is an example of an EVB group file:

```
# Malonaldehyde RS: H9 bonded to O8
-O -i mdin -p mr.top -c mr.crd -o mr.out -r mr.rst -evbin input.mr
# Malonaldehyde PS: H9 bonded to O7
-O -i mdin -p mp.top -c mr.crd -o mp.out -r mp.rst -evbin input.mp
```

Each line corresponds to a diabatic state, and comments are preceded by a # symbol in the first column of a line. Now, it is important to notice in the above example that the starting configurations for both *sander* jobs are the same, although the topology files are different. This constraint guarantees that the system starts in a physically meaningful part of configuration space. Furthermore, it is critical that the atom numbers (delineating the atom locations in the coordinate and parmtop files) are identical among the EVB diabatic states. In Figure 8.1, for example, the atom numbers of the RS and PS malonaldehydes are identical. The only additional flag in the **&cntrl** namelist of the *mdin* file is **ievb**, which has the following values

ievb Flag to run EVB

= *0*	No effect (default)
= *1*	Enable EVB. The value of **imin** specifies if the *sander* calculation is a molecular dynamics (**imin=0**) or an energy minimization (**imin=1**). The variable **evb_dyn** in the **&evb** namelist of the EVB input file refines this choice to specify if the calculation type is on the EVB ground-state surface, on a mapping potential, or on a biased potential.

The argument of the command line flag *-evbin* provides the name of the EVB input file. Corresponding to the above group file example, the inputs for EVB state 1 are provided in the file *input.mr* and those for EVB state 2 are provided in *input.mp*. For the case of constant coupling between the EVB states, the file *input.mr* may look like the following:

```
# Malonaldehyde RS: proton (H9) bound to O8
&evb  nevb = 2,  nbias = 1,  nmorse = 1,  nmodvdw = 1,  ntw_evb = 50,
xch_type    = "constant",
evb_dyn     = "egap_umb",
dia_shift(1)%st = 1, dia_shift(1)%nrg_offset = 0.0,
dia_shift(2)%st = 2, dia_shift(2)%nrg_offset = 0.0,
xch_cnst(1)%ist = 1, xch_cnst(1)%jst = 2,
xch_cnst(1)%xcnst = 12.5,
egap_umb(1)%ist = 1, egap_umb(1)%jst = 2,
egap_umb(1)%k = 0.005, egap_umb(1)%ezero = 0.0,
morsify(1)%iatom = 8, morsify(1)%jatom = 9, morsify(1)%D = 356.570,
morsify(1)%a = 1.046, morsify(1)%r0 = 1.000,
modvdw(1)%iatom = 9, modvdw(1)%jatom = 7,
/
```

and the file *input.mp* may appear as follows:

```
# Malonaldehyde PS: proton (H9) bound to O7
&evb  nevb = 2,  nbias = 1,  nmorse = 1,  nmodvdw = 1,  ntw_evb = 50,
xch_type    = "constant",
evb_dyn     = "egap_umb",
dia_shift(1)%st = 1, dia_shift(1)%nrg_offset = 0.0,
dia_shift(2)%st = 2, dia_shift(2)%nrg_offset = 0.0,
xch_cnst(1)%ist = 1, xch_cnst(1)%jst = 2,
xch_cnst(1)%xcnst = 12.5,
egap_umb(1)%ist = 1, egap_umb(1)%jst = 2,
egap_umb(1)%k = 0.005, egap_umb(1)%ezero = 0.0,
morsify(1)%iatom = 7, morsify(1)%jatom = 9, morsify(1)%D = 356.570,
morsify(1)%a = 1.046, morsify(1)%r0 = 1.000,
modvdw(1)%iatom = 9, modvdw(1)%jatom = 8,
/
```

The above EVB files specify that the system is described by a two-state model, the coupling between the two-states is a constant, and the dynamics is umbrella sampling along an energy gap RC. Since the reactant and product states are identical by symmetry, no adjustments of the relative energies of the diabatic states are performed. The constant value coupling between the two states is parameterized such that the EVB barrier reproduces the ab initio barrier of ~ 3 kcal/mol (RMP2/cc-pVTZ level). Lastly, the standard Amber harmonic bond interactions involving the proton with the donor and acceptor oxygens are replaced by Morse functions and certain van der Waals interactions are excluded.

This parameterization of the EVB surface to provide observables that match either results from high-level quantum chemistry calculations or experimental measurements is the trickiest aspect of the EVB model. However, after the EVB surface has been calibrated, the user has access to reactive chemical dynamics simulation timescales and lengthscales which would be otherwise inaccessible using conventional ab initio MD approaches. The distributed Gaussian EVB framework provides a systematic procedure for computing V_{12} from ab initio data.

Now, let us suppose that the constant coupling prescription does not provide the detailed features needed to describe the reaction pathway. Furthermore, we find that the coupling as a function of the coordinates can be described adequately (from comparison to ab initio data) using a Gaussian functional form. How should one modify the above EVB input files to obtain a more accurate reactive surface? We need to change the **xch_type** variable from **"constant"** to **"gauss"** as well as replace the variable **xch_cnst** by the variable **xch_gauss(:)**, which contains the parameters for the Gaussian functional form. Of course, these parameters need to be optimized to provide the more accurate surface. The modifications to the EVB input files look something like the following,

⋮

~~`xch_type    = "constant",`~~

```
xch_type     = "gauss",
⋮
~~xch_cnst(1)%ist = 1, xch_cnst(1)%jst = 2,~~
~~xch_cnst(1)%xcnst = 12.5,~~
xch_gauss(1)%ist = 1, xch_gauss(1)%jst = 2,
xch_gauss(1)%iatom = 8, xch_gauss(1)%jatom = 7,
xch_gauss(1)%a = 11.0, xch_gauss(1)%sigma = 0.0447,
xch_gauss(1)%r0 = 2.3,
⋮
```

where the cross-through lines have been replaced by those below them. Access to the exponential functional form or the distributed Gaussian approximation to V_{12} entails similar changes to the input files. Please see $AMBERHOME/test/evb for examples.

8.3. Biased sampling

When a reactive event is described by an intrinsic high free energy barrier, molecular dynamics on the EVB ground-state surface will not adequately sample the important transition state region. Under these conditions, chemical reactions are rare events and sampling on the EVB surface effectively reduces to sampling on a diabatic surface. One framework for enhancing the sampling of rare events is through modification of the system Hamiltonian with the addition of biasing potentials. The EVB facility in Amber offers several options for biased sampling: (1) Ariel Warshel's mapping potential approach[242] (2) Dave Case's umbrella sampling on an energy gap RC (3) umbrella sampling on a distance RC and (4) umbrella sampling on a difference of distances RC.

In the mapping potential framework, the system Hamiltonian (and hence, the molecular dynamics) is described by the modified potential

$$V_\lambda = (1-\lambda)V_{ii} + \lambda V_{ff} \tag{8.7}$$

where V_{ii} is the EVB matrix element for the *initial* state and V_{ff} is the EVB matrix element for the *final* state. As the value of the mapping potential parameter λ changes from 0 to 1, the system *evolves* from the initial state to the final state. As an example, for $\lambda = 0.50$, the system Hamiltonian is an equal linear combination of the initial and final states and molecular dynamics sample the region in the vicinity of the transition state. Each mapping potential V_λ samples only a portion of the reaction coordinate. In practice, a series of mapping potentials are used to bias the sampling across the entire range of the RC. The average distribution of the RC for each mapping potential is then *unbiased* and the set of unbiased distributions are combined to give the potential of mean force (PMF) on the EVB ground-state surface. Figure 8.2 shows a PMF for the malonaldehyde intramolecular proton transfer reaction as obtained from 9 mapping potential simulations with λ ranging from 0.10 to 0.90 at 0.10 intervals.

In the umbrella sampling framework, the system Hamiltonian is described by the modified potential

$$\begin{aligned} V^{(n)}_{\text{biased}}(\mathbf{q}) &= V_{\text{el0}}(\mathbf{q}) + V^{(n)}_{\text{umb}}(\mathbf{q}) \\ &= V_{\text{el0}}(\mathbf{q}) + \frac{1}{2}k^{(n)}\left[\text{RC}(\mathbf{q}) - \text{RC}^{(n)}_0\right]^2 \end{aligned} \tag{8.8}$$

where $\mathbf{q}$ is the set of system coordinates, k is the *harmonic force constant* parameter, and $V^{(n)}_{\text{umb}}$ is an umbrella potential that is added to the original system potential V_{el0} (obtained from diagonalization of the EVB matrix) to bias the sampling towards a particular value of the reaction coordinate $\text{RC}^{(n)}_0$. The superscript (n) denotes that a series of biased simulations, each enhancing the sampling of a particular window of the RC, is required to map out the entire PMF. The number of umbrella sampling windows as well as the choice of values for the force constant parameter and the RC equilibrium position will depend ultimately on the nature of the free energy landscape of the system in question.

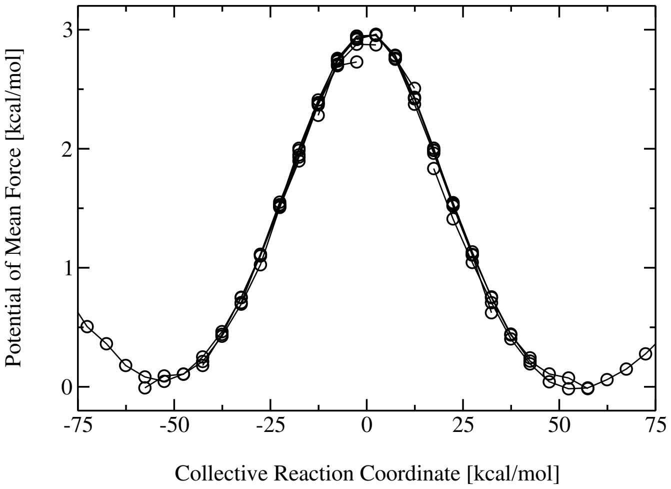


Figure 8.2.: *Potential of mean force along an energy gap RC for the intramolecular proton transfer in malonaldehyde as obtained from a series of mapping potential simulations.*

evb_dyn		dependency
“evb_map”	⇒	**emap(:)**
“egap_umb”	⇒	**egap_umb(:)**
“bond_umb”	⇒	**bond_umb(:)**
“dbonds_umb”	⇒	**dbonds_umb(:)**
“qi_bond_pmf”	⇒	**bond_umb(:)**
“qi_bond_dyn”	⇒	**bond_umb(:)**
“qi_dbonds_pmf”	⇒	**dbonds_umb(:)**
“qi_dbonds_dyn”	⇒	**dbonds_umb(:)**

Table 8.2.: *Derived variable types for EVB.*

Results from the biased samplings then can be unbiased and combined using the weighted histogram analysis method (WHAM)[252–254] to generate the PMF describing chemistry on the physically relevant EVB ground-state potential energy surface, V_{el0}. Figure 8.3 depicts the PMF for the malonaldehyde intramolecular proton transfer that is obtained from 13 umbrella sampling simulations with $\mathrm{RC}_0^{(n)}$ spanning the range -60 kcal/mol to +60 kcal/mol at 10 kcal/mol intervals. The supporting program to generate the PMF from a set of mapping potential or from a set of umbrella sampling simulations can be obtained from the Amber website, *http://ambermd.org*.

Biased sampling is accessed through the **nbias** and **evb_dyn** variables in the EVB input file. The variable **nbias** specifies the number of biasing potentials to include in the system Hamiltonian. Mapping potential dynamics is invoked using the assignment **evb_dyn=“evb_map”**. Biased sampling via umbrella potentials is invoked with the assignment **evb_dyn=“egap_umb”**, **evb_dyn=“bond_umb”** or **evb_dyn=“dbonds_umb”**. Associated with each choice of biased sampling approach is a derived type variable that provides the required parameters, as shown in Table 8.2. Please see Section 8.6 for more details about the variable dependencies.

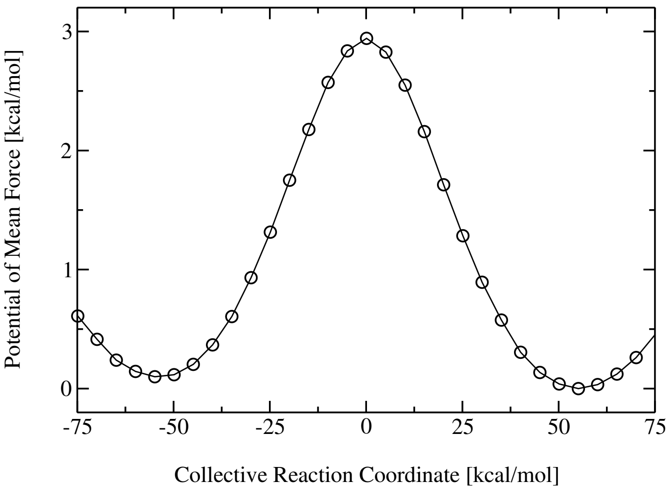


Figure 8.3.: *Potential of mean force for the intramolecular proton transfer in malonaldehyde as obtained from a series of umbrella sampling simulations along an energy gap RC. The distributions of the RC from all the windows are combined using the WHAM procedure.*

8.4. Quantization of nuclear degrees of freedom

The EVB framework provides a computationally practical approximation to the *electronic* surface for modeling chemical reactions involving classical atoms. The full Schrödinger equation, nevertheless, describes not only the electrons but also the nuclei as a wave function. This quantum mechanical description of nuclei is particularly important for capturing the nuclear dispersion of light particles, such as hydrogen. We provide quantization of the nuclear degrees of freedom via coupling of the EVB facility with the Feynman Path-Integral Molecular Dynamics function in Amber [249–251]. The current implementation utilizes the PIMD engine that is built on top of the locally enhanced sampling (LES) infrastructure. As such, the user will need to build the parallel version of LES *sander* in order to utilize EVB/LES-PIMD.

PIMD is invoked using the **ipimd** variable (and associated dependencies) in the **&cntrl** namelist of the *mdin* input file (please consult Section 25.1.2). The requirements for EVB within the EVB/LES-PIMD context are similar to those described for classical EVB but with the coordinate and parmtop files modified for a LES-type calculation, where the number of LES copies correspond to the number of path integral slices. For example, a classical EVB umbrella sampling on a difference of distances RC will have EVB input files similar to the above examples but with the following modifications

```
⋮
evb_dyn    = "dbonds_umb",
⋮
dbonds_umb(1)%iatom = 8, dbonds_umb(1)%jatom = 9, dbonds_umb(1)%katom = 7,
dbonds_umb(1)%k = 100.000, dbonds_umb(1)%ezero = -.20,
⋮
```

EVB/LES-PIMD utilizes these same EVB input files. The EVB group file *evb.grpfile*, however, has been modified to point to the LES coordinate and parmtop files

```
# 32-bead Malonaldehyde RS: H9 bonded to O8
-O -i mdin -p mr_les.top -c mr_les.crd -o mr_les.out -r mr_les.rst \
     -evbin input.mr
# 32-bead Malonaldehyde PS: H9 bonded to O7
-O -i mdin -p mp_les.top -c mr_les.crd -o mp_les.out -r mp_les.rst \
     -evbin input.mp
```

Additionally, the *-nslice <# PIMD slices>* variable must be passed to the *sander* executable:

```
mpirun -np 2 sander.LES.MPI -ng 2 -nslice 32 -groupfile evb.grpfile
```

Here, the atoms of the malonaldehyde system have been replicated into 32 copies using the *addles* utility (see Section 25.1.2) and each of the EVB diabatic states now use the corresponding LES coordinate and parmtop files. Nuclear quantization lowers the free energy barrier due to quantum mechanical effects, such as zero point motion and tunneling. Figure 8.4 compares the PMFs for the malonaldehyde proton transfer reaction along a difference of distances RC from classical EVB and EVB/PIMD umbrella sampling simulations. Currently, only the distance and difference of distances RCs are supported in EVB/PIMD. The energy gap RC is not supported because the theoretical formulation of quantum transition state theory based on an energy gap RC has not yet been worked out.

8.5. Distributed Gaussian EVB

As briefly mentioned in the Introduction to EVB, V_{12} can be fit to high-level electronic structure data using a set of s-, p-, and d-type Gaussians as the fitting basis functions. The current incarnation of DG EVB is limited to two-state gas-phase systems. Current efforts to extend this approach to the condensed phase will provide a practical systematic procedure for constructing a reactive surface from ab initio information. The curious student is encouraged to read the original papers on this method for the theoretical formulation[244, 245]. Here, we only

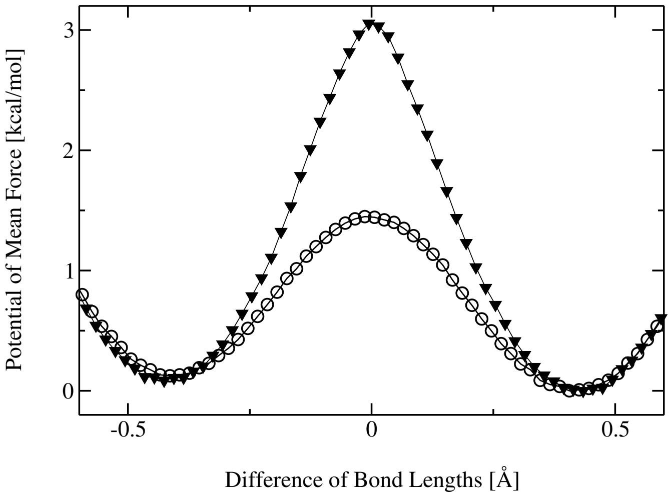


Figure 8.4.: *PMFs as a function of the difference of bond lengths involving the proton with the donor and acceptor oxygens in malonaldehyde. The ▼ curve is from classical EVB, while the ⊙ curve is from EVB/PIMD.*

provide an example of this approach for constructing an ab initio-inspired surface describing the proton transfer reaction in malonaldehyde. All the previously described EVB functionalities are accessible to this method. For example, the key elements of the RS *input.mr* file for biased sampling along a distance RC on the DG EVB surface may look something like the following:

```
⋮
nUFF = 1, nbias = 1,
dia_type = "ab_initio",
xch_type = "dist_gauss",
evb_dyn = "bond_umb",
bond_umb(1)%iatom = 7, bond_umb(1)%jatom = 9,
bond_umb(1)%k = 400.000, bond_umb(1)%ezero = 1.20,
dist_gauss%stype = "no_dihedrals",
dist_gauss%lin_solve = "diis",
dist_gauss%xfile_type = "gaussian_fchk",
ts_xfile(1) = "malonaldehydeTS_35.fchk",
min_xfile(1) = "malonaldehydeR_35.fchk",
min_xfile(2) = "malonaldehydeP_35.fchk",
dgpt_alpha(1) = 0.72,
dgpt_alpha(2) = 0.72,
dgpt_alpha(3) = 0.72,
UFF(1)%iatom = 7, UFF(1)%jatom = 9
⋮
```

These variables are described in Section 8.6. DG EVB is invoked through the **xch_type** variable, with dependencies on **dist_gauss**, **ts_xfile(:)**, **min_xfile(:)**, **dgpt_alpha(:)**, and **UFF(:)**. The ab initio data for the RS minimum are contained in the file **malonaldehydeR_35.fchk**, those for the PS minimum are contained in **malonaldehydeP_35.fchk**, and those for the transition state are contained in **malonaldehydeTS_35.fchk**. These files are in the *Gaussian* [255] formatted checkpoint file format (**gaussian_fchk**). The α parameter [see Eqs. (8.4-8.6)] associated with each of these configuration space points is specified in the variable **dgpt_alpha(:)**. If we wish to include additional ab initio data points along the reaction path, we can specify the file names for those points in the variable **xdg_xfile(:)**. The α parameters associated with these points can be specified in **dgpt_alpha(:)**. It is important to keep in mind that the α parameters are ordered as follows: **dgpt_alpha(**ts_xfile(1), min_xfile(1), min_xfile(2),

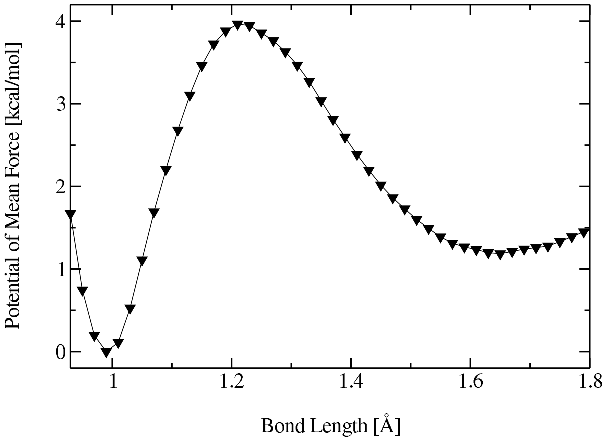


Figure 8.5.: *PMF as a function of the distance between atoms `H9` and `O7` in malonaldehyde. The potential energy surface was constructed from ab initio data using the DG EVB approach.*

xdg_xfile(:)**).** Lastly, the **UFF** variable requests the inclusion of a Universal Force Field [256] repulsive term in V_{11} between the transferred proton (`H9`) and the acceptor (`O7`). The *input.mp* file for the PS V_{22} is identical to the above, but with the **UFF** variable changed to reflect the identity of the acceptor atom from the perspective of the product state topology: **UFF(1)%iatom = 8, UFF(1)%jatom = 9**. In practice, the inclusion of this term to V_{ii} provides a more optimal DG EVB surface for molecular dynamics sampling. Figure 8.5 shows the PMF for shortening the $r_{\mathrm{H9-O7}}$ distance of the malonaldehyde RS from 1.8 Å to 1.0 Å using umbrella sampling of this RC. Note that the PMF is not symmetric because this choice of RC breaks the intrinsic symmetry of the reaction. The difference of distances RC involving atoms `O8`, `H9` and `O7` does provide a symmetric PMF and this is shown in Figure 25.2 within the context of kinetic isotope effect (Section 25.6.5).

8.6. EVB input variables and interdependencies

The variables in the **&evb** namelist of the EVB input file are described below. The style of the input file is similar to the traditional *mdin* used in a *sander* run. Assignment to character type variables need to be encapsulated within quotation marks (for example, **evb_dyn="groundstate"**). Array variables are denoted below by a colon enclosed within parentheses [for example, **dia_shift(:)**]. Derived type variables can be assigned element-wise, i.e., **dia_shift(1)%st = 1, dia_shift(1)%nrg_offset = 0.0**. In the specifications below, the data type of each variable is enclosed in $\{\cdots\}$, while the size of each array variable is enclosed in $[\cdots]$.

ntw_evb {integer}. MD step interval for writing to the EVB output file *evbout*.

nevb {integer}. Number of EVB states. For example, **nevb = 3** specifies that the system is described by a 3×3 EVB matrix in the representation of three diabatic states. The EVB group file will contain three lines of *sander* command line options specifying the *mdin*, coordinate, parmtop, and EVB input files.

nmorse {integer}. Number of Amber harmonic bond interactions that will be changed to a Morse type interaction. Requires additional inputs from the variable **morsify(:)**.

nbias {integer}. Number of biasing potentials to include in the system Hamiltonian. The supported biased sampling approaches include (1) mapping potential, (2) umbrella sampling along an energy gap RC, (3) umbrella sampling along a distance RC, and (4) umbrella sampling along a difference of distances RC. See **evb_dyn** for associated dependencies.

nmodvdw {integer}. Number of van der Waals terms to exclude in the calculation of V_{ii}. Requires additional inputs from the variable **modvdw(:)**.

nuff {integer}. Number of Universal Force Field [256] repulsive terms to include in the harmonic expansion of V_{ii} about the ab initio minimum. Requires additional inputs from the variable **uff(:)**.

xch_type {character*512}. Coupling element type.

= *"constant"* V_{ij} is a constant. Requires additional inputs from the variable **xch_cnst(:)**.

= *"exp"* $V_{ij}(r_{kl}) = A_{ij}\exp\left[-u_{ij}\left(r_{kl}-r_{kl}^{(0,ij)}\right)\right]$. Requires additional inputs from the variable **xch_exp(:)**.

= *"gauss"* $V_{ij}(r_{kl}) = A_{ij}\exp\left[-\frac{1}{\sigma_{ij}^2}\left(r_{kl}-r_{kl}^{(0,ij)}\right)^2\right]$. Requires additional inputs from the variable **xch_gauss(:)**.

= *"dist_gauss"* V_{ij} is described by the Schlegel-Sonnenberg distributed Gaussian approach. Requires additional inputs from the variables **dist_gauss**, **ts_xfile(:)**, **min_xfile(:)**, **xdg_xfile(:)**, **dgpt_alpha(:)**, **uff(:)**.

evb_dyn {character*512}. EVB dynamics type.

= *"groundstate"* Dynamics on the EVB ground-state potential energy surface.

= *"evb_map"* Biased sampling based on Ariel Warshel's mapping potential approach. Requires additional inputs from the variable **emap(:)**.

= *"egap_umb"* Umbrella sampling along an energy gap reaction coordinate. Requires additional inputs from the variable **egap_umb(:)**.

= *"bond_umb"* Umbrella sampling along a distance reaction coordinate. Requires additional inputs from the variable **bond_umb(:)**.

= *"dbonds_umb"* Umbrella sampling along a difference of two distances reaction coordinate. Requires additional inputs from the variable **dbonds_umb(:)**.

= *"qi_bond_pmf"* For generating the QI joint distribution function along the distance RCs of the P and $P/2$ slices (see Section 25.5.2). Requires additional inputs from the variable **bond_umb(:)**.

= *"qi_bond_dyn"* For sampling of the QI f_v, F and G factors with the P and $P/2$ slices constrained to the dividing surfaces along the distance RCs (see Section 25.5.2). Requires additional inputs from the variable **bond_umb(:)**.

= *"qi_dbonds_pmf"* For generating the QI joint distribution function along the difference of distances RCs of the P and $P/2$ slices (see Section 25.5.2). Requires additional inputs from the variable **dbonds_umb(:)**.

= *"qi_dbonds_dyn"* For sampling of the QI f_v, F and G factors with the P and $P/2$ slices constrained to the dividing surfaces along the difference of distances RCs (see Section 25.5.2). Requires additional inputs from the variable **dbonds_umb(:)**.

dia_shift(:) {derived type}, [nevb]. Diabatic state energy shift.

%st	{integer}. Diabatic state index.
%nrg_offset	{real}. Energy offset for EVB state.

xch_cnst(:) {derived type}, [nxch]. Constant coupling. The size of this derived type array is *nxch*, which is calculated internally as $nevb(nevb-1)/2$.

%ist	{integer}. Diabatic state index involved in the coupling.
%jst	{integer}. Diabatic state index involved in the coupling.
%xcnst	{real}. Constant exchange parameter.

xch_exp(:) {derived type}, [nxch]. Parameters for the exponential functional form of the coupling term, $V_{ij}(r_{kl}) = A_{ij}\exp\left[-u_{ij}\left(r_{kl}-r_{kl}^{(0,ij)}\right)\right]$. The size of this derived type array is *nxch*, which is calculated internally as $nevb(nevb-1)/2$.

%ist	{integer}. Diabatic state index involved in the coupling.
%jst	{integer}. Diabatic state index involved in the coupling.
%iatom	{integer}. Index of atom involved in r_{kl}.
%jatom	{integer}. Index of atom involved in r_{kl}.
%a	{real}. A_{ij}.
%u	{real}. u_{ij}.
%r0	{real}. $r_{kl}^{(0,ij)}$.

xch_gauss(:) {derived type}, [nxch]. Parameters for the Gaussian functional form of the coupling term, $V_{ij}(r_{kl}) = A_{ij}\exp\left[-\frac{1}{\sigma_{ij}^2}\left(r_{kl}-r_{kl}^{(0,ij)}\right)^2\right]$. The size of this derived type array is *nxch*, which is calculated internally as $nevb(nevb-1)/2$.

%ist	{integer}. Diabatic state index involved in the coupling.
%jst	{integer}. Diabatic state index involved in the coupling.
%iatom	{integer}. Index of atom involved in r_{kl}.
%jatom	{integer}. Index of atom involved in r_{kl}.
%a	{real}. A_{ij}.
%sigma	{real}. σ_{ij}.
%r0	{real}. $r_{kl}^{(0,ij)}$.

morsify(:) {derived type}, [nmorse]. Parameters used for converting the Amber harmonic bond interactions to the Morse type, $V_{\mathrm{Morse}}(r_{ij}) = D_e\left[1-e^{-\alpha\left(r_{ij}-r_{ij}^{0}\right)}\right]^2$. The components in the derived type are

%iatom	{integer}. Index of atom involved in r_{ij}.

%jatom	{integer}. Index of atom involved in r_{ij}.
%d	{real}. D_e.
%a	{real}. α.
%r0	{real}. r_{ij}^0.

emap(:) {derived type}, [nbias]. Mapping potential parameters required for the function $V_\lambda = (1-\lambda)V_{ii} + \lambda V_{ff}$.

%ist	{integer}. Diabatic state index for the initial state.
%jst	{integer}. Diabatic state index for the final state.
%lambda	{real}. λ.

egap_umb(:) {derived type}, [nbias]. Umbrella potential parameters required for the function $V_{\text{umb}}(\text{RC}) = \frac{1}{2}k\,[\text{RC} - \text{RC}_0]^2$, where $\text{RC} = V_{ii} - V_{ff}$.

%ist	{integer}. Diabatic state index for the initial state.
%jst	{integer}. Diabatic state index for the final state.
%k	{real}. k.
%ezero	{real}. RC_0.

modvdw(:) {derived type}, [nmodvdw]. Exclude the van der Waals interactions between the specified atom pairs.

%iatom	{integer}. Index of atom involved in the non-bonded interaction.
%jatom	{integer}. Index of atom involved in the non-bonded interaction.

bond_umb(:) {derived type}, [nbias]. Umbrella potential parameters for the function $V_{\text{umb}}(\text{RC}) = \frac{1}{2}k\,[\text{RC} - \text{RC}_0]^2$, where $\text{RC} = r_{ij}$.

%iatom	{integer}. Index of atom involved in a distance.
%jatom	{integer}. Index of atom involved in a distance.
%k	{real}. k.
%ezero	{real}. RC_0.

dbonds_umb(:) {derived type}, [nbias]. Umbrella potential parameters for the difference of two distances RC where one of the atoms is common to both distances. $V_{\text{umb}}(\text{RC}) = \frac{1}{2}k\,[\text{RC} - \text{RC}_0]^2$, where $\text{RC} = r_{ij} - r_{kj}$.

%iatom	{integer}. Index of atom involved in a distance.
%jatom	{integer}. Index of the atom common to both distances.
%katom	{integer}. Index of atom involved in a distance.
%k	{real}. k.
%ezero	{real}. RC_0.

out_RCdot {logical}. Output the velocity of a free particle along the RC direction to the file *evbout*.

dist_gauss {derived type}. Schlegel-Sonnenberg distributed Gaussian specifications.

%stype {character*512}. Coordinate selection type. Supported coordinate selection types include *"all_coords"*, *"bonds_only"*,*"no_dihedrals"*,*"react-product"*,*"react-ts-product"*.

%stol {real}. Coordinate selection tolerance for *stype*=*"react-product"* or *stype*=*"react-ts-product"*. For *stype*=*"react-product"*, a particular internal coordinate is used in the DG EVB procedure if the difference between the reactant and product structures is > *stol*. For the case of *stype*=*"react-ts-product"*, the intersection of the selected set of coordinates from *react-ts* > *stol* and *product-ts* > *stol* will be used for the DG EVB procedure.

%xfile_type {character*512}. File type of external ab initio data. Supported file types are *"gaussian_fchk"* and *"EVB"*.

ts_xfile(:) {character*512}, [*]. Name of the file containing the ab initio data corresponding to the transition state.

min_xfile(:) {character*512}, [*]. Name of the file containing the ab initio data corresponding to the minimum, i.e. V_{11} and V_{22}.

xdg_xfile(:) {character*512}, [*]. Name of the file containing the ab initio data corresponding to additional points along the IRC.

dgpt_alpha(:) {real}, [*]. Optimized α parameters associated with the distributed Gaussian data points.

uff(:) {derived type}, [nuff]. Include a UFF repulsive term between the specified atom pairs in the harmonic expansion of V_{ii} about the ab initio minimum.

%iatom {integer}. Index of atom involved in the non-bonded interaction.

%jatom {integer}. Index of atom involved in the non-bonded interaction.

9. sqm: Semi-empirical quantum chemistry

AmberTools contains its own quantum chemistry program, called *sqm*. This is code extracted from the QM/MM portions of *sander*, but is limited to "pure QM" calculations. A principal current use is as a replacement for MOPAC for deriving AM1-bcc charges, but the code is much more general than that. Presently, it is limited to single point calculations and energy minimizations (geometry optimizations) for closed-shell systems. It supports a wide variety of semi-empirical Hamiltonians, including many recent ones. An external electric field generated by a set of point charges can be included for single point calculations. Our plan is to add capabilities to subsequent versions. The major contributors are as follows:

- The original semi-empirical support was written by Ross Walker, Mike Crowley, and Dave Case,[257] based on public-domain MOPAC codes of J.J.P. Stewart.
- DFTB2 (SCC-DFTB) support was written by Gustavo Seabra, Ross Walker and Adrian Roitberg,[258] and is based on earlier work of Marcus Elstner.[259, 260]
- Support for diagonal third-order corrections to SCC-DFTB was written by Gustavo Seabra and Josh Mcclellan.
- DFTB3 was added by Andreas Goetz.
- Various SCF convergence schemes were added by Tim Giese and Darrin York.
- The PM6 Hamiltonian was added by Andreas Goetz and dispersion and hydrogen bond corrections were added by Andreas Goetz and Kyoyeon Park.
- The extension for MNDO type Hamiltonians to support d orbitals was written by Tai-Sung Lee, Darrin York and Andreas Goetz.
- The charge-dependent exchange-dispersion corrections of vdW interactions[261] was contributed by Tai-Sung Lee, Tim Giese, and Darrin York.
- Support for reading user-defined parameters for NDDO methods was added by Tai-Sung Lee and Darrin York.

The DFTB/DFTB2 code was originally based on the DFT/DYLAX code by Marcus Elstner *et al.*, but has since been extensively re-written and optimized. The DFTB3 implementation is an extension of this code.

9.1. Available Hamiltonians

Available MNDO-type semi-empirical Hamiltonians are PM3,[262] AM1,[263] RM1,[264] MNDO,[265] PDDG/PM3,[266] PDDG/MNDO,[266] PM3CARB1,[267], PM3-MAIS[268, 269], MNDO/d[270–272], AM1/d (Mg from AM1/d[273] and H, O, and P from AM1/d-PhoT[274]) and PM6[275].

Also available is the density functional theory-based tight-binding (DFTB) Hamiltonian[258, 276, 277] and its self-consistent-charge version with Taylor expansion up to second order (SCC-DFTB or DFTB2)[259] and third-order (DFTB3)[278]. If you use the mio-1-1 parameters for DFTB2, you can add an empirical correction for dispersion effects[279] and calculate CM3 charges[280] (both only for elements H, C, N, O, S, P). Diagonal third-order corrections are available for DFTB2[281] with mio-1-1 parameters but it is recommended to perform full DFTB3 simulations instead. Neither dispersion corrections nor halogen corrections are implemented for DFTB3.

The elements supported by each QM method are:

- MNDO: H, Li, Be, B, C, N, O, F, Al, Si, P, S, Cl, Zn, Ge, Br, Cd, Sn, I, Hg, Pb
- MNDO/d: H, Li, Be, B, C, N, O, F, Na, Mg, Al, Si, P, S, Cl, Zn, Ge, Br, Sn, I, Hg, Pb
- AM1: H, C, N, O, F, Al, Si, P, S, Cl, Zn, Ge, Br, I, Hg
- AM1/d: H, C, N, O, F, Mg, Al, Si, P, S, Cl, Zn, Ge, Br, I, Hg
- PM3: H, Be, C, N, O, F, Mg, Al, Si, P, S, Cl, Zn, Ga, Ge, As, Se, Br, Cd, In, Sn, Sb, Te, I, Hg, Tl, Pb, Bi
- PDDG/PM3: H, C, N, O, F, Si, P, S, Cl, Br, I
- PDDG/MNDO: H, C, N, O, F, Cl, Br, I
- RM1: H, C, N, O, P, S, F, Cl, Br, I
- PM3CARB1: H, C, O
- PM3-MAIS: H, O, Cl
- PM6: H, He, Li, Be, B, C, N, O, F, Ne, Na, Mg, Al, Si, P, S, Cl, Ar, K, Ca, Sc, Ti, V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Ga, Ge, As, Se, Br, Kr, Rb, Sr, Y, Zr, Nb, Mo, Tc, Ru, Rh, Pd, Ag, Cd, In, Sn, Sb, Te, I, Xe, Cs, Ba, La, Lu, Hf, Ta, W, Re, Os, Ir, Pt, Au, Hg, Tl, Pb, Bi
- DFTB/DFTB2/DFTB3: (Any atoms for which parameters are available from *www.dftb.org*)

The PM6 implementation has not been extensively tested for all available elements. Please check your results carefully, possibly by comparison to other codes that implement PM6, if transition metal elements are present. SCF convergence may be more difficult to achieve for transition metal elements with partially filled valence shells.

If the PM6 Hamiltonian is used in a QM/MM simulation with *sander* using electrostatic embedding (see Section 10) or if an electric field of external point charges is used, then the electrostatic interactions between QM and MM atoms are modeled using the MNDO type core repulsion function for interactions between QM and MM atoms. Parameters for the exponents α of the QM atoms are taken from PM3 (a default value of five is used for the exponents α of the MM atoms as is the case for MNDO, AM1 and PM3). Since PM3 does not have parameters for all elements that are supported by PM6, the missing exponents were defined in an ad hoc manner (see the source code in $AMBERHOME/AmberToosl/src/sqm/qm2_parameters.F90, variable alp_pm6). The magnitude of the coefficients α is probably not critical for the accuracy of QM/MM calculations but this should be tested on a case by case basis. This does not affect QM calculations with *sqm*.

9.1.1. DFTB parameter files

In order to use DFTB2 or DFTB3 (*qm_theory=DFTB*2 or *DFTB3*) a set of integral parameter files is required. The mio-1-1 parameter files for DFTB2 and 3ob-3-1 parameter files are distributed with Amber under a Creative Commons Attribution-ShareAlike 4.0 International License, see http://creativecommons.org/licenses/by-sa/4.0/. The parameters were obtained from the website *www.dftb.*org on February 22, 2017. You may want to check if there are any updates to the parameters. If you perform DFTB simulations, in addition to Amber please cite the publications describing the QM/MM and DFTB implementations as well as following references for the DFTB parameters:

When using DFTB2 with mio-1-1 and following elements:

- O, N, C, H: M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, Th. Frauenheim, S. Suhai, G. Seifert, *Phys. Rev. B* **58** (1998) 7260.
- S: T. A. Niehaus, M. Elstner, Th. Fruanheim, S. Suhai, *J. Molec. Struct. (THEOCHEM)* **541** (2001) 185.
- P: M. Gaus, Q. Cui, M. Elstner, *J. Chem. Theory Comput.* **7** (2011) 931-948.

When using DFTB3 with 3ob-3-1 and following elements:

- O, N, C, H: M. Gaus, A. Goez, M. Elstner, *J. Chem. Theory Comput.* **9** (2013) 338-354.
- P, S: M. Gaus, X. Lu, M. Elstner, Q. Cui, *J. Chem. Theory Comput.* **10** (2014) 1518-1537.
- Mg, Zn: X. Lu, M. Gaus, M. Elstner, Q. Cui, *J. Phys. Chem. B* **119** (2015) 1062-1082.
- Na, F, K, Ca, Cl, Br, I: M. Kubillus, T. Kubar, M. Gaus, J. Rezac, M. Esltner, *J. Chem. Theory Comput.* **11** (2015) 332-342.

Additional parameter files can be obtained from the website *www.dftb.org*. By default it is assumed that DFTB2 uses the mio-1-1 parameter set and DFTB3 the 3ob-3-1 parameter set and that the corresponding files with extension *.skf* reside in the directories *$AMBERHOME/dat/slko/mio-1-1* and *$AMBERHOME/dat/slko/3ob-3-1*. If you want to use other parameter sets and/or put the parameter files in other directories then you have to specify the location in the input file (keyword *dftb_slko_path,* see section 9.3 for details).

Following parameter files for use with DFTB2 and the mio-1-1 parameter set are also distributed with AmberTools: Dispersion parameters for H, C, N, O, P and S are available in the file *$AMBERHOME/dat/slko/mio-1-1/DISPERSION.INP_ONCHSP*, CM3 parameters for the same atoms are in the file *$AMBERHOME/dat/slko/mio-1-1/CM3_PARAMETERS.DAT* file, and two parametrizations for diagonal third-order SCC-DFTB terms (SCC-DFTB-PA and SCC-DFTB-PR) are in the files *DFTB_3RD_ORDER_PA.DAT* and *DFTB_3RD_ORDER_PR.DAT*, both located in the same directory.

9.2. Dispersion and hydrogen bond correction

An empirical dispersion and hydrogen bonding correction is implemented for the MNDO type Hamiltonians AM1 and PM6[282]. The empirical dispersion correction follows the formalism for DFT-D[283] and consists of a physically sound r^{-6} term that is damped at short distances to avoid the short-range repulsion which can be written as

$$E_{\mathrm{d}is} = -s_6 \sum_{ij} f_{\mathrm{d}amp}(r_{ij}, R^0_{ij}) C_{6,ij} r_{ij}^{-6}, \tag{9.1}$$

where r_{ij} is the distance between two atoms i and j, R^0_{ij} is the equilibrium van der Waals (vdW) separation derived from the atomic vdW radii, $C_{6,ij}$ the dispersion coefficient, and s_6 a general scaling factor. The damping function is given as

$$f_{\mathrm{d}amp}(r_{ij}, R^0_{ij}) = \left[1 + \exp\left(-\alpha \frac{r_{ij}}{s_R R^0_{ij}} - 1\right)\right]^{-1}. \tag{9.2}$$

Bondi vdW radii[284] are used and for a pair of unlike atoms we have

$$R^0_{ij} = \frac{R^{0\,3}_{ii} + R^{0\,3}_{jj}}{R^{0\,2}_{ii} + R^{0\,2}_{jj}}. \tag{9.3}$$

For the C_6 coefficients the following equation is used,

$$C_{6,ij} = 2\frac{(C^2_{6,ii} C^2_{6,jj} N_{\mathrm{e}ff,i} N_{\mathrm{e}ff,j})^{1/3}}{(C_{6,ii} N^2_{\mathrm{e}ff,j})^{1/3} + (C_{6,jj} N^2_{\mathrm{e}ff,i})^{1/3}}, \tag{9.4}$$

where the Slater-Kirkwood effective number of electrons $N_{\mathrm{e}ff,i}$ and the C_6 coefficients can easily be found in the literature[283].

An empirical hydrogen bonding correction[282] that is transferable among different semiempirical Hamiltonians and has been parametrized for use with the dispersion correction described above is also available. This correction does not make the assumption of a specific acceptor/hydrogen/donor binding situation. Instead it considers the hydrogen bond as a charge-independent atom- atom term between two atoms capable of serving as an acceptor or donor (for example, O, N) and weights this by a function that accounts for the steric arrangement of the two atoms and the favorable positioning of a hydrogen atom inbetween. A damping function corrects for long- and short-range behavior,

$$E_{\mathrm{H}-bond} = \frac{C_{AB}}{r_{AB}^2} f_{geom} f_{\mathrm{d}amp}, \tag{9.5}$$

$$f_{geom} = \cos(\theta_A)^2 \cos(\phi_A)^2 \cos(\psi_A)^2 \cos(\phi_B)^2 \cos(\phi_B)^2 \cos(\psi_B)^2 f_{\mathrm{b}ond}, \tag{9.6}$$

$$f_{\mathrm{b}ond} = 1 - \frac{1}{1 + \exp[-60(r_{\mathrm{X}H}/1.2 - 1)]}, \tag{9.7}$$

$$f_{\mathrm{d}amp} = \left(\frac{1}{1 + \exp[-100(r_{AB}/2.4 - 1)]} \right) \left(1 - \frac{1}{1 + \exp[-10(r_{\mathrm{A}B}/7.0 - 1)]} \right), \tag{9.8}$$

$$C_{\mathrm{A}B} = \frac{C_{\mathrm{A}} + C_{\mathrm{B}}}{2}. \tag{9.9}$$

Here, C_{A} and C_{B} are the atomic hydrogen bonding correction parameters and the (torsion) angles in the function f_{geom} are defined similarly to an earlier hydrogen bond correction[285].

The hydrogen bond correction can be used both for single point energy calculations or geometry optimizations with SQM and for molecular dynamics simulations with SANDER. However, we do not recommend the use for molecular dynamics at present since cutoffs needed to be implemented for the calculation of f_{geom} of equation (9.6). This and some other conditional evaluations give rise to discontinuities in the potential energy surface and thus make this method unattractive for MD simulations.

9.3. Usage

The *sqm* program uses the following simple command line:

```
sqm [-O] -i <input-file> -o <output-file>
```

mdin is the default input-file name, and *mdout* is the default output-file name.As in other Amber programs, the “-O” flag allows the program to over-write the output file.

An example input file for running a simple minimization is shown here:

```
Run semi-empirical minimization
 &qmmm
  qm_theory='AM1',   qmcharge=0,
 /
   6    CG       -1.9590        0.1020        0.7950
   6   CD1       -1.2490        0.6020       -0.3030
   6   CD2       -2.0710        0.8650        1.9630
   6   CE1       -0.6460        1.8630       -0.2340
   6    C6       -1.4720        2.1290        2.0310
   6    CZ       -0.7590        2.6270        0.9340
   1   HE2       -1.5580        2.7190        2.9310
  16   S15       -2.7820        0.3650        3.0600
   1   H19       -3.5410        0.9790        3.2740
   1   H29       -0.7870       -0.0430       -0.9380
   1   H30        0.3730        2.0450       -0.7840
   1   H31       -0.0920        3.5780        0.7810
   1   H32       -2.3790       -0.9160        0.9010
```

The *&qmmm* namelist contains variables that allow you to control the options used. Following that is one line per atom, giving the atomic number, atom name, and Cartesian coordinates (free format). The variables in the *&qmmm* namelist are these:

qm_theory	Level of theory to use for the QM region of the simulation (Hamiltonian). Default is to use the semi-empirical Hamiltonian PM3. Options are AM1, RM1, MNDO, PM3-PDDG, MNDO-PDDG, PM3-CARB1, MNDO/d (same as MNDOD), AM1/d (same as AM1D), PM6, DFTB2 (same as DFTB), and

DFTB3. The dispersion correction can be switched on for AM1 and PM6 by choosing AM1-D* and PM6-D, respectively. The dispersion and hydrogen bond correction will be applied for AM1-DH+ and PM6-DH+.

dftb_slko_path Path to the DFTB Slater-Koster parameter files. Defaults to '$AMBERHOME/dat/slko/mio-1-1/' for DFTB2 and '$AMBERHOME/dat/slko/3ob-3-1/' for DFTB3. You can specify a different directory here, which is assumed to be a subdirectory of '$AMBERHOME/dat/slko/' unless you specify an absolute path.

dftb_disper Flag turning on (1) or off (0) the use of a dispersion correction to the DFTB2 energy (only for mio-1-1 parameters). Requires *qm_theory=DFTB2*. It is assumed that you have the file DISPERSION.INP_ONCHSP in your $AMBERHOME/dat/slko/mio-1-1 directory. This file must be downloaded from the website www.dftb.org, as described in the beginning of this chapter. Only available for elements H, C, O, N, P, S. (Default = 0)

dftb_3rd_order Third order diagonal corrections to DFTB2 with mio-1-1 parameters. Default=" (the empty string which means no third order correction).

= 'PA' Use the SCC-DFTB-PA parametrization, which was developed for proton affinities. The parameters will be read from the *$AMBERHOME/dat/slko/DFTB_3RD_ORDER_PA.DAT* file.

= 'PR' Use the SCC-DFTB-PR parametrization, which was developed for phosphate hydrolysis reactions. The parameters will be read from the *$AMBERHOME/dat/slko/DFTB_3RD_ORDER_PR.DAT* file.

= 'READ' Parameters will be read from the *mdin* file, in a separate "dftb_3rd_order" namelist, which must have the same format as the files above.

= '*filename*' Parameters will be read from the file specified by *filename*, in the "dftb_3rd_order" namelist, which must have the same format as the files above.

dftb_chg Flag to choose the type of charges to report when doing a DFTB calculation.

= 0 (default) - Print Mulliken charges.

= 2 Print CM3 charges. Only available for DFTB2 with mio-1-1 parameters for elements H, C, N, O, S and P.

dftb_telec Electronic temperature, in K, used to accelerate SCC convergence in DFTB calculations. The electronic temperature affects the Fermi distribution promoting some HOMO/LUMO mixing, which can accelerate the convergence in difficult cases. In most cases, a low *telec* (around 100K) is enough. Should be used only when necessary, and the results checked carefully. Default: 0.0K

dftb_maxiter Maximum number of SCC iterations before resetting Broyden in DFTB calculations. (default: 70)

qmcharge Charge on the QM system in electron units (must be an integer). (Default = 0)

spin Multiplicity of the QM system. Currently only singlet calculations are possible and so the default value of 1 is the only available option. Note that this option is ignored by DFTB/SCC-DFTB, which allows only ground state calculations. In this case, the spin state will be calculated from the number of electrons and orbital occupancy.

qmqmdx Flag for whether to use analytical or numerical derivatives of the semiemprical electron repulsion integrals. The default (and recommended) option is to use ANALYTICAL QM-QM derivatives.

= 1 (default) - Use analytical derivatives for QM-QM forces.

= 2 Use numerical derivatives for QM-QM forces. Note: the numerical derivative code has not been optimised as aggressively as the analytical code and as such is significantly slower. Numerical derivatives are intended mainly for testing purposes.

verbosity Controls the verbosity of QM/MM related output. *Warning:* Values of 2 or higher will produce a *lot* of output.

= 0 (default) - only minimal information is printed - Initial QM geometry and link atom positions as well as the SCF energy at every ntpr steps.

= 1 Print SCF energy at every step to many more significant figures than usual. Also print the number of SCF cycles needed on each step.

= 2 As 1 and also print info about memory reallocations, number of pairs per QM atom, QM core - QM core energy, QM core - MM atom energy, and total energy.

= 3 As 2 and also print SCF convergence information at every step.

= 4 As 3 and also print forces on the QM atoms due to the SCF calculation and the coordinates of the link atoms at every step.

= 5 As 4 and also print all of the info in kJ/mol as well as kcal/mol.

tight_p_conv Controls the tightness of the convergence criteria on the density matrix in the SCF.

=0 (default) - loose convergence on the density matrix (or Mulliken charges, in case of a SCC-DFTB calculation). SCF will converge if the energy is converged to within scfconv and the largest change in the density matrix is within 0.05*sqrt(scfconv).

= 1 Tight convergence on density(or Mulliken charges, in case of a SCC-DFTB calculation). Use same convergence (scfconv) for both energy and density (charges) in SCF. Note: in the SCC-DFTB case, this option can lead to instabilities.

scfconv Controls the convergence criteria for the SCF calculation, in kcal/mol. In order to conserve energy in a dynamics simulation with no thermostat it is often necessary to use a convergence criterion of 1.0d-9 or tighter. Note, the tighter the convergence the longer the calculation will take. Values tighter than 1.0d-11 are not recommended as these can lead to oscillations in the SCF, due to limitations in machine precision, that can lead to convergence failures. Default is 1.0d-8 kcal/mol. Minimum usable value is 1.0d-14.

pseudo_diag Controls the use of 'fast' pseudo diagonalisations in the SCF routine. By default the code will attempt to do pseudo diagonalisations whenever possible. However, if you experience convergence problems then turning this option off may help. Not available for DFTB/SCC-DFTB.

= 0 Always do full diagonalisation.

= 1 Do pseudo diagonalisations when possible (default).

pseudo_diag_criteria Float controlling criteria used to determine if a pseudo diagonalisation can be done. If the difference in the largest density matrix element between two SCF iterations is less than this criteria then a pseudo diagonalisation can be done. This is really a tuning parameter designed for expert use only. Most users should have no cause to adjust this parameter. (Not applicable to DFTB/SCC-DFTB calculations.) Default = 0.05

diag_routine Controls which diagonalization routine will be used during the SCF procedure. This is an advanced option to fine-tune performance which has negligible effect on energies (and generally little effect on geometries in the case of SQM energy minimizations). The speed of each diagonalizer is a function of the number and type of QM atoms as well as the LAPACK library that the program was linked to. As such there is not always an obvious choice to obtain the best performance. The simplest option is to set diag_routine = 0 in which case the program will test each diagonalizer in turn, including the pseudo diagonalizer, and select the one that gives optimum performance. As of AmberTools 15 diag_routine = 0 is the default for both SQM and QMMM in Sander. Not available for DFTB/SCC-DFTB.

= 0 Automatically select the fastest routine (default).

= 1 Use internal diagonalization routine.

= 2 Use lapack dspev.

= 3 Use lapack dspevd.

= 4 Use lapack dspevx.

= 5 Use lapack dsyev.

= 6 Use lapack dsyevd.

= 7 Use lapack dsyevr.

printcharges **= 0** Don't print any info about QM atom charges to the output file (default)

= 1 Print Mulliken QM atom charges to output file every *ntpr* steps.

print_eigenvalues Controls printing of MO eigenvalues.

= 0 Do not print MO eigenvalues

= 1 Print MO eigenvalues at the end of a single point calculation or geometry optimization (default)

= 2 Print MO eigenvalues at the end of every SCF cycle (only NDDO methods, not DFTB)

= 3 Print MO eigenvalues during each step of the SCF cycle (only NDDO methods, not DFTB)

qxd Flag to turn on (=.true.) or off (=.false., default) the charge-dependent exchange-dispersion corrections of vdW interactions[261].

parameter_file **=** 'PARAM.FILE' Read user-defined parameters from the file 'PARAM.FILE'. The first three space-separated entries (case insensitive) of each line will be interpreted as a user-modified parameter in the sequence of *parameter name, element name,* and *value*. For example, a line contains "USS Cl -111.6139480D0 " will cause the USS parameter of the Cl element changed to -111.6139480. A line beginning with "END" will stop the reading. This function currently only works for MNDO, AM1, PM3, MNDO/d, and AM1/d. Also, when new nuclear core-core parameters (FN, in PM3, AM1, and AM1/d) are re-defined, the number of FNN parameter sets (NUM_FN) also needs to be defined. For example, if FN*n*3 (n = 1, 2, or 3) is defined, then NUM_FN needs to be set to 3 or 4.

peptide_corr **= 0** Don't apply MM correction to peptide linkages. (default)

= 1 Apply a MM correction to peptide linkages. This correction is of the form $E_{scf} = E_{scf} + h_{type}(i_{type})\sin^2\phi$, where ϕ is the dihedral angle of the H-N-C-O linkage and h_{type} is a constant dependent on the Hamiltonian used. (Recommended, except for DFTB/SCC-DFTB.)

itrmax Integer specifying the maximum number of SCF iterations to perform before assuming that convergence has failed. Default is 1000. Typically higher values will not do much good since if the SCF hasn't converged after 1000 steps it is unlikely to. If the convergence criteria have not been met after itrmax steps the SCF will stop and the minimisation will proceed with the gradient at itrmax. Hence if you have a system which does not converge well you can set itrmax smaller so less time is wasted before assuming the system won't converge. In this way you may be able to get out of a bad geometry quite quickly. Once in a better geometry SCF convergence should improve.

maxcyc Maximum number of minimization cycles to allow, using the *xmin* minimizer (see Section 39.5) with the TNCG method. Default is 9999. Single point calculations can be done with *maxcyc* = 0.

ntpr Print the progress of the minimization every *ntpr* steps; default is 10.

grms_tol Terminate minimization when the gradient falls below this value; default is 0.02

ndiis_attempts Controls the number of iterations that DIIS (direct inversion of the iterative subspace) extrapolations will be attempted. Not available for DFTB/SCC-DFTB. The SCF does not even begin to exhaust its attempts at using DIIS extrapolations until the end of iteration 100. Therefore, for example, if ndiis_attempts=50, then DIIS extrapolations would be performed at end of iterations 100 to 150. The purpose of not performing DIIS extrapolations before iteration 100 is because the existing code base performs quite well for most molecules; however, if convergence is not met after 100 iterations, then it is presumed that further iterations will not yield SCF convergence without doing something different, i.e., DIIS. Thus, the implementation of DIIS in SQM is a mechanism to try and force SCF convergence for molecules that are otherwise difficult to converge. Default 0. Maximum 1000. Minimum 0. Note that DIIS will automatically turn itself on for 100 attempts at the end of iteration 800 even if you did not explicitly set ndiis_attempts to a nonzero value. This is done as a final effort to achieve convergence.

ndiis_matrices Controls the number of matrices used in the DIIS extrapolation. Including only one matrix is the same as not performing an extrapolation. Including an excessive number of matrices may require a large amount of memory. Not available for DFTB/SCC-DFTB. Default 6. Minimum 1. Maximum 20.

vshift Controls level shifting (only NDDO methods, not DFTB). Virtual orbitals can be shifted up by vshift (in eV) to improve SCF convergence in cases with small HOMO/LUMO gap. Default 0.0 (no level shift).

errconv SCF tolerance on the maximum absolute value of the error matrix, i.e., the commutator of the Fock matrix with the density matrix. The value has units of hartree. The default value of errconv is sufficiently large to effectively remove this tolerance from the SCF convergence criteria. Not available for DFTB/SCC-DFTB. Default 1.d-1. Minimum 1.d-16. Maximum 1.d0.

qmmm_int When running QM calculations in the sqm program, an electric field of external point charges can be added. In this way, the electrostatic effect outside of the QM region can be modeled, making the calculation a simplified QM/MM calculation without QM/MM vdW's contribution. Like QM/MM calculations (see Section 10), the method to couple QM and MM electrostatic interactions for external charges and semiempirical Hamiltonians can be specified via the *qmmm_int* namelist variable.

The current implementation limits use of external charges to only single point energy calculations. To run such a calculation, an additional field, which begins with #EXCHARGES and ends with #END, is required to specify the external point charges in the input. Each external point charge must include atomic number, atom name, X, Y, Z coordinates and the charge in units of the electron charge. An example input looks like:

```
single point energy calculation (adenine), with external charges (thymine)
&qmmm
 qm_theory = 'PM3',
 qmcharge = 0,
 maxcyc = 0,
 qmmm_int = 1,
/
7  N   1.0716177  -0.0765366   1.9391390
1  H   0.0586915  -0.0423765   2.0039181
1  H   1.6443796  -0.0347395   2.7619159
6  C   1.6739638  -0.0357766   0.7424316
7  N   0.9350155  -0.0279801  -0.3788916
6  C   1.5490760   0.0012569  -1.5808009
1  H   0.8794435   0.0050260  -2.4315709
7  N   2.8531510   0.0258031  -1.8409596
6  C   3.5646109   0.0195446  -0.7059872
6  C   3.0747955  -0.0094480   0.5994562
7  N   4.0885824  -0.0054429   1.5289786
6  C   5.1829921   0.0253971   0.7872176
1  H   6.1882591   0.0375542   1.1738824
7  N   4.9294871   0.0412404  -0.5567274
```

```
1  H   5.6035368   0.0648755  -1.3036811
#EXCHARGESwill be
6  C  -4.7106131   0.0413373   2.1738637  -0.03140
1  H  -4.4267056   0.9186178   2.7530256   0.06002
1  H  -4.4439282  -0.8302573   2.7695655   0.05964
1  H  -5.7883971   0.0505530   2.0247280   0.03694
6  C  -3.9917387   0.0219348   0.8663338  -0.25383
6  C  -4.6136833   0.0169051  -0.3336520   0.03789
1  H  -5.6909220   0.0269347  -0.4227183   0.16330
7  N  -3.9211729  -0.0009646  -1.5163659  -0.47122
1  H  -4.4017172  -0.0036078  -2.4004924   0.35466
6  C  -2.5395897  -0.0149474  -1.5962357   0.80253
8  O  -1.9416783  -0.0291878  -2.6573783  -0.63850
7  N  -1.9256484  -0.0110593  -0.3638948  -0.58423
1  H  -0.8838255  -0.0216168  -0.3784269   0.35404
6  C  -2.5361367   0.0074651   0.8766724   0.71625
8  O  -1.8674730   0.0112093   1.9120833  -0.60609
#END
```

10. QM/MM calculations

Sander supports the option of describing part of the system quantum mechanically in an approach known as a hybrid (or coupled potential) QM/MM simulation. Semi-empirical neglect of diatomic overlap (NDDO)-type and density functional tight binding (DFTB) Hamiltonians are supported natively by *sander* and the basic documentation (e.g. what Hamiltonians are implemented, description of the input parameters) can be found in Chapter 9. Here we limit our description to those features that are unique to the QM/MM interface implemented in *sander*. More advanced Hamiltonians based on *ab initio* wave function theory (WFT) and density functional theory (DFT) are supported via an interface to external QM software packages the use of which is described in section 10.2.

The built-in semi-empirical QM/MM support was written by Ross Walker and Mike Crowley, [257] based originally on public-domain MOPAC codes of J.J.P. Stewart. The QM/MM generalized Born implementation uses the model described by Pellegrini and Field[286] while regular QM/MM Ewald support is based on the work of Nam *et al.*[287] with QM/MM PME support based on the work of Walker *et al.*[257]. SCC-DFTB support was written by Gustavo Seabra, Ross Walker and Adrian Roitberg,[258] and is based on earlier work of Marcus Elstner.[259, 260] Support for third-order SCC-DFTB was written by Gustavo Seabra and Josh McClellan.

10.1. Built-in semiempirical NDDO methods and SCC-DFTB

When running a QM/MM simulation in *sander* the system is partitioned into two regions, a QM region consisting of the atoms defined by either the *qmmask* or *iqmatoms* keyword, and a MM region consisting of all the atoms that are not part of the QM region. For a typical protein simulation in explicit solvent the number of MM atoms will be much greater than the number of QM atoms. Either region can contain zero atoms, giving either a pure QM simulation or a standard classical simulation. For periodic simulations, the quantum region must be *compact*, so that the extent (or diameter) of the QM region (in any direction) plus twice the QM/MM cutoff must be less than the box size. Hence, you can define an "active site" to be the QM region, but in most cases could not ask that all cysteine residues (for example) be quantum objects. The restrictions are looser for non-periodic (gas-phase or generalized Born) simulations, but the codes are written and tested for the case of a single, compact quantum region.

The partitioned system is characterized by an effective Hamiltonian which operates on the system's wavefunction Ψ, which is dependent on the position of the MM and QM nuclei, to yield the system energy E_{eff}:

$$H_{eff}\Psi(x_e,x_{QM},x_{MM}) = E_{eff}(x_{QM},x_{MM})\Psi(x_e,x_{QM},x_{MM}) \tag{10.1}$$

The effective Hamiltonian consists of three components - one for the QM region, one for the MM region and a term that describes the interaction of the QM and MM regions, implying that likewise the energy of the system can be divided into three components. If the total energy of the system is re-written as the expectation value of H_{eff} then the MM term can be removed from the integral since it is independent of the position of the electrons:

$$E_{eff} = \langle\Psi|H_{QM} + H_{QM/MM}|\Psi\rangle + E_{MM} \tag{10.2}$$

In the QM/MM implementation in *sander*, E_{MM} is calculated classically from the MM atom positions using the Amber or CHARMM force field equation and parameters, whereas H_{QM} is evaluated using the chosen QM method.

The interaction term $H_{QM/MM}$ is more complicated. By default, *sander* uses an electrostatic embedding scheme (also referred to as additive scheme) in which the interaction of the MM point charges with the electrons of the QM system as well as the interaction between the MM point charges and the QM nuclei (atomic cores for semi-empirical methods) is explicitly taken into account. In other words, the MM region polarizes the QM electron density. For the case where there are no covalent bonds between the atoms of the QM and MM regions the interaction Hamiltonian is thus the sum of an electrostatic term and a Lennard-Jones (VDW) term and can be written as

$$H_{QM/MM} = \sum_q \sum_m \left[Q_m h_{electron}(x_e, x_{MM}) - Q_m Z_q h_{core}(x_{QM}, x_{MM}) + \left(\frac{A}{r_{qm}^{12}} - \frac{B}{r_{qm}^{6}} \right) \right] \tag{10.3}$$

where the subscripts e, m and q refer to the electrons, the MM nuclei and the QM nuclei respectively. Here Q_m is the charge on MM atom m, Z_q is the core charge (nucleus minus core electrons) on QM atom q, r_{qm} is the distance between atoms q and m, and A and B are Lennard-Jones interaction parameters. For systems that have covalent bonds between the QM and MM regions, the situation is more complicated, as discussed later.

A more approximate form of the interaction term $H_{QM/MM}$ is referred to as mechanical embedding (or subtractive QM/MM scheme). In this case the interactions between the QM and the MM region are obtained within the same classical approximation that is used for the MM region, that is

$$H_{QM/MM} = \sum_q \sum_m \left[\frac{Q_m Q_q}{r_{qm}} + \left(\frac{A}{r_{qm}^{12}} - \frac{B}{r_{qm}^{6}} \right) \right] \tag{10.4}$$

where Q_q is the classical MM point charge assigned to an atom in the QM region. Mechanical embedding is useful to impose steric constraints on the embedded QM system, however, the electron density is not polarized by the MM environment. An additional complication of this approach is that the point charges that are assigned to the atoms in the QM region have to represent the electrostatic potential of the QM region during the whole course of a QM/MM simulation.

If one evaluates the expectation values in Eq. 10.2 over a single determinant built from molecular orbitals

$$\phi_i = \sum_j c_{ij} \chi_j \tag{10.5}$$

where the c_{ij} are molecular orbital coefficients and the χ_j are atomic basis functions, the total energy depends upon the c_{ij} and on the positions x_{MM} and x_{QM} of the atoms. The energy is obtained by setting $\partial E_{eff}/\partial c_{ij}$ to zero which leads to a self-consistent (SCF) procedure to determine the c_{ij}, (with a modified Fock matrix that contains the electric field arising from the MM charges in the case of electrostatic embedding). Once the energy is known, the forces on the atoms can be obtained by taking the derivative of the energy expression with respect to the positions of the QM and MM atoms.

The main subtlety that arises in the case of electrostatic embedding is that, for a periodic system, there are formally an infinite number of QM/MM interactions; even for a non-periodic system, the (finite) number of such interactions may be prohibitively large. These problems are addressed in a manner analogous to that used for pure MM systems: a PME approach is used for periodic systems, and a (large) cutoff may be invoked for non-periodic systems. Some details are discussed below.

10.1.1. The QM/MM interface and link atoms

The sections above dealt with situations where there are no covalent bonds between the QM and MM regions. In many protein simulations, however, it is necessary to have the QM/MM boundary cut covalent bonds, and a number of additional approximations have to be made. There are a variety of approaches to this problem, including hybrid orbitals, capping potentials, and explicit link atoms. The last option is the method available in *sander*.

There are a number of ways to implement a link atom approach that deal with the way the link atom is positioned, the way the forces on the link atom are propagated, and the way non-bonding interactions around the link atom are treated. Each time an energy or gradient calculation is to be done, the link atom coordinates are re-generated from the current coordinates of the QM and MM atoms making up the QM-MM covalent pair. The link atom is placed along the bond vector joining the QM and MM atom, at a distance d_{L-QM} from the QM atom. By default d_{L-QM} is set to the equilibrium distance of a methyl C-H atom pair (1.09 Å) but this can be set in the input file. The default link atom type is hydrogen, but this can also be specified as an input.

Since the link atom position is a function of the coordinates of the "real" atoms, it does not introduce any new degrees of freedom into the system. The chain rule is used to re-write forces on the link atom itself in terms of forces on the two real atoms that define its position. This is analogous to the way in which "extra points" or "lone-pairs" are handled in MM force fields.

The remaining details of how the QM-MM boundary is treated are as follows: for the interactions surrounding the link atom, the MM bond term between the QM and MM atoms is calculated classically using the classical force field parameters, as are any angle or dihedral terms that include at least one MM atom. The Lennard-Jones interactions between QM-MM atom pairs are calculated in the same way as described in the section above with exclusion of 1-2 and 1-3 interactions and scaling of 1-4 interactions. What remains is to specify the electrostatic interactions between QM and MM atoms around the region of the link atom.

A number of different schemes have been proposed for handling link-atom electrostatics. Many of these have been tested or calibrated on (small) gas-phase systems, but such testing can neglect some considerations that are very important for more extended, condensed-phase simulations. In choosing our scheme, we wanted to ensure that the total charge of the system is rigorously conserved (at the correct value) during an MD simulation. Further, we strove to have the Mulliken charge on the link atom (and the polarity of its bond to the nearest QM atom) adopt reasonable values and to exhibit only small fluctuations during MD simulations. Link atoms interact with the MM field in exactly the same way as regular QM atoms. That is they interact with the electrostatic field due to all the MM atoms that are within the cutoff, with the exception of the MM link pair atoms (MM atoms that are bound directly to QM atoms). VDW interactions are not calculated for link atoms. These are calculated between all real QM atoms and all MM atoms, including the MM link pair atoms. For Generalized Born simulations the effective Born radii for the link atoms are calculated using the intrinsic radii for the MM link pair atoms that they are replacing.

In the case of electrostatic embedding the atoms that make up the QM region (including the MM link pair atom) have their charges from the prmtop file essentially replaced with Mulliken charges. Hence it is important to consider the issue of charge conservation. The QM region (including the link atoms) by definition must have an integer charge. This is defined by the &qmmm namelist variable *qmcharge*. If the MM atoms (including the MM link pair atoms) that make up the QM region have prmtop charges that sum to the value of *qmcharge* then there is no problem. If not, there are two options for dealing with this charge, defined by the namelist variable *adjust_q*. A value of 1 will distribute the difference in charge equally between the nearest *nlink* MM atoms to the MM link pair atoms. A value of 2 will distribute this charge equally over all of the MM atoms in the simulation (excluding MM link pair atoms).

10.1.2. A reformulated QM/MM interface for PM3

In the current version of Amber, a reformulated QM-MM core-charge potential (denoted as PM3/MM*) has been implemented. This reformulated potential scales the interaction between a QM core and a MM charge for the purpose of better description of the geometry and energy at the QM-MM interface:[288]

$$E^{core}_{QM/MM} = Z_a q_m \left(s_a s_a, s_m s_m\right) \left[1 + \frac{|q_m|}{q_m} \cdot \left(-e^{-f_1^a \cdot R_{am}} + e^{-f_2^a \cdot R_{am}}\right)\right] \tag{10.6}$$

where Z_a is the effective core charge of QM atom a, q_m is the partial charge on MM atom m, s_a is an s orbital on the QM atom, s_m is a notional s orbital on the MM atom, R_{am} is the QM-MM interatomic distance, and f_1^a and f_2^a are exponential scale factors which depend on the QM atom only. Optimal values for f_1^a and f_2^a were determined based on the PM3 Hamiltonian, and are available for H, C, N and O atoms (so the QM region is limited to these four atoms; but the MM region is not restricted). Application of this reformulated potential shows improved prediction of geometry and interaction energy at the QM-MM interface for hydrogen bonded small molecule complexes typical of biomolecular interactions, without significantly impacting the modeling of other interaction types, such as dispersion dominant complexes.[288] In a QM/MM calculation, giving *qmmm_int=3* along with *qm_theory=PM3* will invoke this potential.

Based on PM3/MM*, further developments to the semi-empirical QM/MM coupling method have been introduced – PM3/MMX2 (*qmmm_int*=4 and *qm_theory*=PM3) – which shares the same QM core-MM charge equation with the PM3/MM* model. In addition, a QM parameter, ρ_{mm}, is introduced to each type of QM atoms in order to "fine-tune" the QM electron-MM charge interaction (Eq. 10.7). Although ρ_{mm} is a parameter for QM atom, the subscript *mm* emphasizes that it is a MM-related property (eqn 3.xx). Parameters are currently avaiable for H, C, N, O and S QM atoms (manuscript in preparation).

$$E_{QM/MM}^{electron} = -q_m(\mu_a\nu_a, s_m s_m) = \sum_{\ell_a}\sum_{\ell_m}\left[M_{l_a k}^{a} M_{l_m k}^{m}\right] \tag{10.7}$$

where

$$\left[M_{l_a k}^{a} M_{l_m k}^{m}\right] = \frac{e^2}{2^{l_a+l_m}} \sum_{i=1}^{2^{l_a}} \sum_{j=1}^{l} \left[r_{ij}^2 + \left(\rho_{l_a+}^{a}\rho_{mm}^{a}\right)^2\right]^{-1/2} \tag{10.8}$$

10.1.3. Generalized Born implicit solvent

The implementation of Generalized Born (GB) for QM/MM calculations is based on the method described by Pellegrini and Field.[286] Here, the total energy is taken to be E_{eff} from Eq. 10.2 plus E_{gb} from Eq. 4.2. In E_{gb}, charges on the QM atoms are taken to be the Mulliken charges determined from the quantum calculation; hence these charges depend upon the molecular orbital coefficients c_{ij} as well as the positions of the atoms.

As with conventional QM/MM simulations, one then solves for the c_{ij} by setting $\partial E_{eff}/\partial c_{ij} = 0$. This leads to a set of SCF equations with a Fock matrix modified not only by the presence of MM atoms (as in "ordinary" QM/MM simulations), but also modified by the presence of the GB polarization terms. Once self-consistency is achieved, the resulting Mulliken charges can be used in the ordinary way to compute the GB contribution to the total energy and forces on the atoms.

10.1.4. Ewald and PME

The support for long range electrostatics in QM/MM calculations using electrostatic embedding is based on a modification of the Nam, Gao and York Ewald method for QM/MM calculations.[287] This approach works in a similar fashion to GB in that Mulliken charges are used to represent long range interactions. Within the cut off, interactions between QM and MM atoms are calculated using a full multipole treatment. Outside of the cut off the interaction is based on pairwise point charge interactions. For semiempirical NDDO-type methods this leads to a slight discontinuity at the QM/MM cut off boundary and thus a small energy drift during QM/MM MD simulations in the NVE ensemble. This energy drift can be avoided by using a switching function at the cutoff (see below).

The implementation in Ref [287] uses an Ewald sum for both QM/QM and QM/MM electrostatic interactions. This can be expensive for large MM regions, and thus *sander* uses a modification of this method by Walker and Crowley[257] that uses a PME model (rather than an Ewald sum) for QM/MM interactions. This is controlled by the *qm_pme* variable discussed below.

When running QM/MM Ewald or PME simulations in *sander*, if QM multipoles are involved in QM-MM interactions (NDDO methods), a discontinuity in the QM-MM electrostatic potential occurs at the cut-off distance due to the sudden change in the potential function (the difference between Eqs. 10.9 and 10.10) , thus resulting in energy conservation problems in the simulation.

$$E_{QM/MM}^{r<cutoff} = -q_m(\mu_a\nu_a, s_m s_m) + Z_a q_m(s_a s_a, s_m s_m)(1+scale) \tag{10.9}$$

$$E_{QM/MM}^{r>cutoff} = \frac{q_m(Z_a - \sum c_{\mu\mu})}{r} \tag{10.10}$$

This problem can be avoided by applying a switching function to smoothly connect the two different potentials. The QM/MM electrostatic potential using a switching function can thus be written as:

$$E_{QM/MM} = E_{QM/MM}^{r<cutoff}\, s(r) + E_{QM/MM}^{r>cutoff}\,(1-s(r))$$

The switching function can be turned on or off via the *&qmmm* namelist variable *qmmm_switch,* for details see section 10.1.6 below.

10.1.5. Hints for running successful QM/MM calculations

Required Parameters and Prmtop Creation

QM/MM calculations without link atoms require mass, charges, van der Waals and GB radii in the *prmtop* file. All bonds, angles, and dihedrals parameters involving QM atoms are neglected. In the case of electrostatic embedding the charges are also neglected. (Note that when SHAKE is applied to the QM reg, the bonds are constrained to the ideal MM values, even when these are part of a QM region; hence, for this case, it is important to have correct bond parameters in the QM region.) The simplest general prescription for setting things up is to use *antechamber* and *LEaP* to create a reference force field, since "placeholders" are required in the *prmtop* file even for things that will be neglected. This also allows you to run comparison simulations between pure MM and QM/MM simulations, which can be helpful if problems are encountered in the QM/MM calculations.

The use of *antechamber* to construct a pure MM reference system is even more useful when there are link atoms, since here MM parameters for bonds, angles and dihedrals that cross the QM/MM boundary are also needed.

Choosing the QM region

There are no good universal rules here. Generally, one might want to have as large a QM region as possible, but having more than 80-100 atoms in the QM region will lead to simulations that are very expensive. One should also remember that for many features of conformational analysis, a good MM force field may be better than a semiempirical or DFTB quantum description. In choosing the QM/MM boundary, it is better to cut non-polar bonds (such as C-C single bonds) than to cut unsaturated or polar bonds. Link atoms are not placed between bonds to hydrogen. Thus cutting across a C-H bond will NOT give you a link atom across that bond. (This is not currently tested for in the code and so it is up to the user to avoid such a situation.) Furthermore, link atoms are restricted to one per MM link pair atom. This is tested for during the detection of link atoms and an error is generated if this requirement is violated. This would seem to be a sensible policy otherwise you could have two link atoms too close together. See the comments in *qm_link_atoms.f* for a more in-depth discussion of this limitation.

Choice of electrostatic cutoff

The implementation of the non-bonded cut off in QM/MM simulations is slightly different than in regular MM simulations. The cut off between MM-MM atoms is still handled in a pairwise fashion. However, for QM atoms any MM atom that is within *qmcut* of ANY QM atom is included in the interaction list for all QM atoms. This means that the value of *qmcut* essentially specifies a shell around the QM region rather than a spherical shell around each individual QM atom. Ideally the cut off should be large enough that the energy as a function of the cutoff has converged. For non-periodic, generalized Born simulations, a cutoff of 15 to 20 Å seems sufficient in some tests. (Remember that long-range electrostatic interactions are reduced by a factor of 80 from their gas-phase counterparts, and by more if a nonzero salt concentration is used.) For periodic simulations, the cutoff only serves to divide the interactions between "direct" and "reciprocal" parts; as with pure MM calculations, a cutoff of 8 or 9 Å is sufficient here.

Parallel simulations

The built-in QM/MM implementation currently supports execution in parallel via the message passing interface (MPI), however, the implementation is not fully parallel. At present all parts of the QM simulation are parallel except the density matrix build and the matrix diagonalisation. For small QM systems these two operations do not take a large percentage of time and so acceptable scaling can be seen to around 8 CPU cores (depending on type of CPU and/or interconnect speed between compute nodes). For large QM systems the matrix diagonalisation time will dominate and so the scaling will not be as good. In this case it may be beneficial to choose a LAPACK diagonalization routine in combination with a threaded library such as the Intel Math Kernel Library (MKL). For details on how to choose the diagonalization routine see Section 9.3. The number of threads to be used for the diagonalization is set via an environment variable of the operating system (typically OMP_NUM_THREADS).

10.1.6. General QM/MM &qmmm Namelist Variables

An example input file for running a simple QM/MM MD simulation is shown here:

```
&cntrl
 imin=0, nstlim=10000,          ! Perform MD for 10,000 steps
 dt=0.002,                      ! 2 fs time step
 ntt=1, tempi=0.1, temp0=300.0, ! Berendsen temperature control
 ntb=1,                         ! Constant volume periodic boundaries
 ntf=2, ntc=2,                  ! Shake hydrogen atoms
 cut=8.0,                       ! 8 angstrom classical non-bond cut off
 ifqnt=1                        ! Switch on QM/MM coupled potential
/
&qmmm
 qmmask=':753',    ! Residue 753 should be treated using QM
 qmcharge=-2,      ! Charge on QM region is -2
 qm_theory='PM3', ! Use the PM3 semi-empirical Hamiltonian
 qmcut=8.0         ! Use 8 angstrom cut off for QM region
/
```

The *&qmmm* namelist contains variables that allow you to control the options used for a QM/MM simulation. This namelist must be present when running QM/MM simulations and at the very least must contain either the *iqmatoms* or *qmmask* variable which define the region to be treated quantum mechanically. If *ifqnt* is set to zero then the contents of this namelist are ignored.

For the QM region definition specify one of either *iqmatoms* or *qmmask*. Link atoms will be added automatically along bonds (as defined in the prmtop file) that cross the QM/MM boundary.

iqmatoms	comma-separated integer list containing the atom numbers (from the prmtop file) of the atoms to be treated quantum mechanically.
qmmask	Mask specifying the quantum atoms. E.g. :1-2, = residues 1 and 2. See mask documentation for more info.
qmcut	Specifies the size of the electrostatic cutoff in Angstroms for QM/MM electrostatic interactions. By default this is the same as the value of cut chosen for the classical region, and the default generally does not need to be changed. Any classical atom that is within qmcut of *any* QM atom is included in the pair list. For PME calculations, this parameter just affects the division of forces between direct and reciprocal space. *Note:* this option only effects the electrostatic interactions between the QM and MM regions. Within the QM region all QM atoms see all other QM atoms regardless of their separation. QM-MM van der Waals interactions are handled classically, using the cutoff value specified by *cut*.
qm_ewald	This option specifies how long range electrostatics for the QM region should be treated. **= 0** Use a real-space cutoff for QM-QM and QM-MM long range interactions. In this situation QM atoms do not see their images and QM-MM interactions are truncated at the cutoff. This is the default for non-periodic simulations. **= 1** (default) Use PME or an Ewald sum to calculate long range QM-QM and QM-MM electrostatic interactions. This is the default when running QM/MM with periodic boundaries and PME. **= 2** This option is similar to option 1 but instead of varying the charges on the QM images as the central QM region changes the QM image charges are fixed at the Mulliken charges obtained from the previous MD step. This approach offers a speed improvement over *qm_ewald=1*, since the SCF typically converges in fewer steps, with only a minor loss of accuracy in the long range electrostatics. This option has not been extensively tested, although it becomes increasingly accurate as the box size gets larger.

kmaxqx,y,z Specifies the maximum number of kspace vectors to use in the x, y and z dimensions respectively when doing an Ewald sum for QM-MM and QM-QM interactions. Higher values give greater accuracy in the long range electrostatics but at the expense of calculation speed. The default value of 8 should be optimal for most systems.

ksqmaxq Specifies the maximum number of K squared values for the spherical cut off in reciprocal space when doing a QM-MM Ewald sum. The default value of 100 should be optimal for most systems.

qm_pme Specifies whether a PME approach or regular Ewald approach should be used for calculating the long range QM-QM and QM-MM electrostatic interactions.

- **= 0** Use a regular Ewald approach for calculating QM-MM and QM-QM long range electrostatics. Note this option is often much slower than a pme approach and typically requires very large amounts of memory. It is recommended only for testing purposes.
- **= 1** (default) Use a QM compatible PME approach to calculate the long range QM-MM electrostatic energies and forces and the long range QM-QM forces. The long range QM-QM energies are calculated using a regular Ewald approach.

qmmm_switch Specifies whether a switching function shall be used at the cutoff for long range electrostatics (applies only to NDDO methods). The lower and higher boundaries of the switching function are user definable, see *r_switch_lo* and *r_switch_hi*.

- **= 0** (default). Do not use a switching function. This leads to slight discontinuities in the potential at the cut off and thus an energy drift in NVE simulations.
- **= 1** Use a switching function. See also variables *r_switch_hi* and *r_switch_lo*.

r_switch_hi Specifies the upper boundary of the switching function in Å (see *qmmm_switch*). Defaults to *qmcut.*

r_switch_lo Specifies lower boundary of the switching function in Å (see *qmmm_switch*). Defaults to *r_switch_hi* - 2.

qmgb Specifies how the QM region should be treated with generalized Born.

- **= 2** (default) As described above, the electrostatic and "polarization" fields from the MM charges and the exterior dielectric (respectively) are included in the Fock matrix for the QM Hamiltonian.
- **= 3** This is intended as a debugging option and should only be used for single point calculations. With this option the GB energy is calculated using the Mulliken charges as with option 2 above but the fock matrix is NOT modified by the GB field. This allows one to calculate what the GB energy would be for a given structure using the gas phase quantum charges. When combined with a simulation using *qmgb=2*, this allows the strain energy from solvation to be calculated.

qm_theory Level of theory to use for the QM region of the simulation. (Hamiltonian). Default is to use the semi-empirical hamiltonian PM3. See the *Section 9.3* for details.

qmmm_int Controls the way in which QM/MM interactions are handled in the direct space QMMM sum. This controls only the electrostatic interactions. VDW interactions are always calculated classically using the standard 6-12 potential. Note: with the exception of qmmm_int=0 DFTB calculations (qm_theory=DFTB) always use a simple mulliken charge - resp charge interaction and the value of qmmm_int has no influence.

- **= 0** This turns off all electrostatic interaction between QM and MM atoms in the direct space sum. Note QM-MM VDW interactions will still be calculated classically.
- **= 1** (default) QM-MM interactions in direct space are calculated in the same way for all of the various semi-empirical hamiltonians. The interaction is calculated in an analogous way to the the core-core interaction between QM atoms. The MM resp charges are included in the one electron hamiltonian so that QMcore-MMResp and QMelectron-MMResp interactions are calculated.

= 2 This is the same as for 1 above except that when AM1, PM3 or Hamiltonians derived from these are in use the extra Gaussian terms that are introduced in these methods to improve the core-core repulsion term in QM-QM interactions are also included for the QM-MM interactions. This is the equivalent to the QM-MM interaction method used in CHARMM and DYNAMO. It tends to slightly reduce the repulsion between QM and MM atoms at small distances. For distances above approximately 3.5 angstroms it makes almost no difference.

= 3 Using this along with *qm_theory=PM3* invokes a reformulated QM core-MM charge potential at the QM-MM interface (Eq. 10.6). Current parametrization limits the QM region to H, C, N and O atoms only; MM region is not restricted.[288]

= 4 Currently not in use.

= 5 Mechanical embedding: The electrostatic interaction between QM and MM atoms is treated on the same level as within the MM region using the classical force field point charges also for the QM atoms. The electronic Hamiltonian does not contain the field generated by the MM region point charges and thus the electron density is not polarized by the MM environment. Does not work with GB. Not extensively tested in presence of link atoms.

qmshake Controls whether SHAKE is applied to QM atoms. Using SHAKE on the QM region will allow you to use larger time steps such as 2 fs with NTC=2. If, however, you expect bonds involving hydrogen to be broken during a simulation you should not SHAKE for the QM region. WARNING: the SHAKE routine uses the equilibrium bond lengths as specified in the prmtop file to reset the atom positions. Thus while bond force constants and equilibrium distances are not used in the energy calculation for QM atoms the equilibrium bond length is still required if QM SHAKE is on.

= 0 Do not shake QM H atoms.

= 1 Shake QM H atoms if SHAKE is turned on (NTC>1) (default).

printdipole Controls whether the dipole moment shall be printed every *ntpr* steps.

= 0 Do not print the dipole moment (default).

= 1 Print the dipole moment of the QM region.

= 2 Print the total dipole moment of the QM and MM region.

writepdb **= 0** Do not write a PDB file of the selected QM region. (default).

= 1 Write a PDB file of the QM region. This option is designed to act as an aid to the user to allow easy checking of what atoms were included in the QM region. When this option is set a crude PDB file of the atoms in the QM region will be written on the very first step to the file *qmmm_region.pdb*.

vsolv Controls whether solvent molecules shall be included into the QM region (requires settings in the *&vsolv* namelist; see also section 10.3 on adaptive solvent QM/MM simulations, in particular the namelist information in section 10.3.2.2).

= 0 Do not include solvent molecules into the QM region (default).

= 1 Include solvent molecules via simple solvent switching (requires *&vsolv* namelist).

= 2 Adaptive solvent QM/MM with fixed number of solvent molecules in A and T regions (requires *&vsolv* and *&adqmmm* namelists).

= 3 Adaptive solvent QM/MM with fixed size of A and T regions (requires *&vsolv* and *&adqmmm* namelists).

In addition to the above parameters, the following variables may be set, as described in Section 9.3:
qm_theory, dftb_disper, dftb_3rd_order , dftb_chg , dftb_telec , dftb_maxiter , qmcharge, spin, qmqmdx, verbosity, tight_p_conv, scfconv, pseudo_diag, pseudo_diag_criteria, diag_routine, printcharges, qxd, parameter_file, peptide_corr, and itrmax.

10.1.7. Link Atom Specific QM/MM &qmmm Namelist Variables

The following options go in the *&qmmm* namelist and control the link atom behaviour.

lnk_dis Distance in Å from the QM atom to its link atom. Currently all link atoms must be placed at the same distance. A negative value of lnk_dis specifies that the link atom should be placed directly on top of the MM link pair atom. In this case the distance of the link atom from the QM region changes as a function of time and the actual value of lnk_dis is ignored. Additionally this means that not all link atoms will be placed at the same distance. Negative values of lnk_dis will work with regular link atoms, such as hydrogen, but are really intended for use with pseudo atom / capping approaches. Default = 1.09Å.

lnk_method This defines how classical valence terms that cross the QM/MM boundary are dealt with.

=1 (Default) in this case any bond, angle or dihedral that involves at least one MM atom, including the MM link pair atom is included. This means the following (where QM = QM atom, MM = MM atom, MML = MM link pair atom.):

Bonds = MM-MM, MM-MML, MML-QM

Angles = MM-MM-MM, MM-MM-MML, MM-MML-QM, MML-QM-QM

Dihedrals = MM-MM-MM-MM, MM-MM-MM-MML, MM-MM-MM-MML-QM, MM-MML-QM-QM, MML-QM-QM-QM

=2 Only include valence terms that include a full MM atom, that is, count the MM link pair atom as effectively being a QM atom. This option is designed to be used in conjunction with a pseudo atom / capping type approach where the link atom is parameterized specifically to behave like a uni-valent version of the MM atom it replaces. This option gives the following interactions:

Bonds = MM-MM, MM-MML

Angles = MM-MM-MM, MM-MM-MML, MM-MML-QM

Dihedrals = MM-MM-MM-MM, MM-MM-MM-MML, MM-MM-MML-QM, MM-MML-QM-QM

lnk_atomic_no The atomic number of the link atoms. This selects what element the link atoms are to be. Default = 1 (Hydrogen). Note this must be an integer and an atomic number supported by the chosen QM theory.

adjust_q This controls how charge is conserved during a QMMM calculation involving link atoms. When the QM region is defined the QM atoms and any MM atoms involved in link bonds have their RESP charges zeroed. If the sum of these RESP charges does not exactly match the value of *qmcharge* then the total charge of the system will not be correct.

= 0 No adjustment of the charge is done.

= 1 The charge correction is applied to the nearest *nlink* MM atoms to MM atoms that form link pairs. Typically this will be any MM atom that is bonded to a MM link pair atom (a MM atom that is part of a QM-MM bond). This results in the total charge of QM+QMlink+MM equaling the original total system charge from the prmtop file. Requires natom-nquant-nlink >= nlink and nlink>0.

= 2 (default) - This option is similar to option 1 but instead the correction is divided among all MM atoms (except for those adjacent to link atoms). As with option 1 this ensures that the total charge of the QM/MM system is the same as that in the *prmtop* file. Requires natom-nquant-nlink >= nlink.

10.1.8. Charge-dependent exchange-dispersion corrections of vdW interactions

The *sqm* program provides a new charge-dependent energy model consisting of van der Waals (vdW) and polarization interactions between the quantum mechanical (QM) and molecular mechanical (MM) regions in a combined QM/MM calculation. vdW interactions are commonly treated using empirical Lennard-Jones (L-J) potentials, whose parameters are often chosen based on the QM atom type (e.g., based on hybridization or specific covalent bonding environment). This strategy for determination of QM/MM nonbonding interactions becomes tedious to parametrize and lacks robust transferability. Problems occur in the study of chemical reactions where the "atom type" is a complex function of the reaction coordinate. This is particularly problematic for reactions, where atoms or localized functional groups undergo changes in charge state and hybridization.

In *sqm*, this charge-dependent energy model was implemented based on a scaled overlap model for repulsive exchange and attractive dispersion interactions that is a function of atomic charge. The model is chemically significant since it properly correlates atomic size, softness, polarizability, and dispersion terms with minimal one-body parameters that are functions of the atomic charge[261].

This “Charge-dependent exchange-dispersion corrections of vdW interactions” can be invoked by the “qxd=.true.” switch in the *&qmmm* namelist. Note that this model currently does not have any effect on pure quantum calculations through *sqm*, the qxd correction is only added to QM/MM interactions in *sander*. The default values of qxd parameters are set to reproduce the regular L-J interactions of typical atom types (HC for H, C* for C, N for N, OW for O, and parameters for F and Cl are optimized[261]) when the charge dependence parameters are zero. There are eight qxd parameters (symbols used in the reference[261] are indicated in the parentheses): qxd_s (s), qxd_z0 ($\zeta(0)$), qxd_zq (ζ_q), qxd_d0 (α_1), qxd_dq ($3 \times B$), qxd_q0 (α_2), qxd_qq ($3 \times B$), and qxd_neff ($N_{eff}(0)$). All parameters can be modified through external user-defined parameter files (see the usage of 'parameter_file' in Section 9.3).

10.2. Interface for *ab initio* and DFT methods

In addition to the built-in semi-empirical methods *sander* also supports QM/MM simulations with *ab initio* wave function theory (WFT) and density functional theory (DFT) potentials via an interface to external QM software packages[289]. The implementation makes use of the existing QM/MM infrastructure that has been developed earlier for the semi-empirical methods. Thus, much of AMBER’s previous QM/MM functionality such as the user-friendly link atom approach are available and the implementation remains simple and transparent to use without any significant additional steps in the simulation setup as compared to semi-empirical QM/MM simulations. At present the interface supports several well-known and widely used QM software packages. Mechanical embedding is available for

- ADF (Amsterdam Density Functional) [290, 291]
- GAMESS-US [292, 293]
- NWChem [294]

Mechanical and electrostatic embedding is available for

- Gaussian [255]
- Orca [295]
- Q-Chem[296][296]
- TeraChem [297]
- MRCC [298, 299]

While ADF, Gaussian, Q-Chem and TeraChem are commercial programs, GAMESS-US, NWChem, Orca and MRCC are available at no cost for academic research. The interface has been written in a modular fashion and is easily extensible to support other QM software packages. It is our intention to keep adding support for other software packages. If you are interested in interfacing a specific program, please do not hesitate to contact us.

The interface was developed by Andreas Goetz (SDSC, UCSD) with help of Matthew Clark (SDSC) and support by Ross Walker (SDSC, UCSD). Thanks are due to Christine Isborn and Todd Martinez (Stanford University) for modifications to the TeraChem code to support this interface, to Mark Williamson (University of Cambridge) for an initial version of the module that supports NWChem and Bence Hégely for contributing code that supports MRCC. If you make use of this interface, please cite the following work:

- A. W. Götz, M. A. Clark, R. C. Walker, *An extensible interface for QM/MM molecular dynamics simulations with AMBER*, J. Comput. Chem. **35**, 95-108 (2014), DOI: 10.1002/jcc.23444

If you are using the interface with the TeraChem code, please cite in addition the following work:

- C. M. Isborn, A. W. Götz, M. A. Clark, R. C. Walker, T. J. Martínez, *Electronic Absorption Spectra from MM and ab initio QM/MM Molecular Dynamics: Environmental Effects on the Absorption Spectrum of Photoactive Yellow Protein*, J. Chem. Theory Comput. **8**, 5092-5106 (2012), DOI: 10.1021/ct3006826

If you are using the interface with the MRCC code, please cite in addition the following work:

- B. Hégely, F. Bogár, G. G. Ferenczy, M. Kállay, *A QM/MM program for calculations with frozen localized orbitals based on the Huzinaga equation*, Theoret. Chem. Acc. **134**, 132 (2015), DOI: 10.1007/978-3-662-49825-5_16

Access to QM methods not available within Amber is also possible via the Amber interface to the PUPIL simulation framework. For details, see refs. 300, 301. In what follows we will describe the new interface that is native to *sander*.

10.2.1. Theory

As described in section 10.1, the Hamiltonian of a system that is partitioned into a QM region that is treated with WFT and a classical region that is treated with MM consists of three components and the energy associated with this Hamiltonian is obtained as the corresponding expectation value

$$E = \langle \Psi | \mathscr{H}_{\mathrm{QM}} + \mathscr{H}_{\mathrm{QM}/MM} | \Psi \rangle + E_{\mathrm{MM}}. \tag{10.11}$$

A QM/MM calculation therefore requires not only to choose the WFT used in the QM region and the MM model used for the MM region, but in addition also the form of the QM/MM Hamiltonian which describes the interaction between the quantum and the classical region. The most simple approach is to neglect any electronic coupling between the QM and the MM system and include only the classical non-bonded van der Waals (vdW) and electrostatic interactions between the QM and the MM atoms. This is useful to impose steric constraints on the embedded QM system and commonly referred to as mechanical embedding. In most cases, however, it is better to allow for an explicit polarization of the QM system due to the presence of the point charges on the MM atoms. This is referred to as electronic embedding and the resulting interaction energy becomes

$$\begin{aligned} E^{electronic}_{QM/MM} &= \sum_{A \in \mathrm{MM}} \int \rho(\mathbf{r}) \frac{Q_A}{|\mathbf{r} - \mathbf{R}_A|} d\mathbf{r} + \sum_{A \in \mathrm{QM}, B \in MM} \frac{Z_A Q_B}{R_{AB}} \\ &+ \sum_{A \in QM, B \in MM} \varepsilon_{AB} \left[\left(\frac{\sigma_{AB}}{R_{AB}} \right)^{12} - \left(\frac{\sigma_{AB}}{R_{AB}} \right)^{6} \right]. \end{aligned} \tag{10.12}$$

This QM/MM energy expression also holds for DFT and the terms represent, in order, the electrostatic interaction between the QM electron density and the MM point charges, the electrostatic interation between the QM point charge nuclei and the MM point charges, and the van der Waals repulsion between the QM and MM atoms.

The forces acting on an atom A in a QM/MM calculation are given in terms of derivatives of the total energy expression (10.11) with respect to the Cartesian coordinates of the atom,

$$\mathbf{F}_A = -\nabla_A E_{\mathrm{QM}} - \nabla_A E_{\mathrm{QM}/MM} - \nabla_A E_{\mathrm{MM}}, \tag{10.13}$$

where $\nabla_A = \partial/\partial \mathbf{R}_A = (\partial/\partial R^x_A, \partial/\partial R^y_A, \partial/\partial R^z_A)$. If a QM and an MM program are coupled for QM/MM calculations, the QM program will calculate the QM forces $-\nabla_A E_{\mathrm{Q}M}$ acting on QM atoms and the MM program the MM forces $-\nabla_A E_{\mathrm{M}M}$ acting on the MM atoms. All that remains, is to calculate the forces acting on QM and MM atoms due to the QM/MM interaction energy, $-\nabla_A E_{\mathrm{Q}M/MM}$. For mechanical embedding this will be entirely handled by the MM program. For electronic embedding the forces are given as

$$\begin{aligned}\nabla_A E^{electronic}_{\mathrm{QM}/MM} &= Z_A \sum_{B\in \mathrm{MM}} \frac{Q_B(\mathbf{R}_A-\mathbf{R}_B)}{R^3_{AB}} + \sum_{B\in \mathrm{MM}} \int \frac{\partial\rho(\mathbf{r})}{\partial \mathbf{R}_A} \frac{Q_B}{|\mathbf{r}-\mathbf{R}_B|}\, d\mathbf{r} + \sum_{B\in \mathrm{MM}} \nabla_A V^{\mathrm{LJ}}_{AB} \\ &= -Z_A \mathbf{E}_{\mathrm{M}M}(\mathbf{R}_A) - \int \rho(\mathbf{r}) \mathbf{E}_{\mathrm{M}M}(\mathbf{r})\, d\mathbf{r} + \sum_{B\in \mathrm{MM}} \nabla_A V^{\mathrm{LJ}}_{AB}\end{aligned} \tag{10.14}$$

for the derivatives with respect to the positions of the QM atoms A where $\mathbf{E}_{\mathrm{M}M}$ is the electric field generated by the MM point charges and and V^{LJ}_{AB} is the Lennard-Jones potential from (10.12) and

$$\begin{aligned}\nabla_B E^{electronic}_{\mathrm{QM}/MM} &= Q_B \sum_{A\in \mathrm{Q}M} \frac{Z_A(\mathbf{R}_B-\mathbf{R}_A)}{R^3_{AB}} + \int \rho(\mathbf{r}) \frac{Q_B(\mathbf{R}_B-\mathbf{r})}{|\mathbf{r}-\mathbf{R}_B|^3}\, d\mathbf{r} + \nabla_B E^{\mathrm{me}chanic}_{\mathrm{QM}/MM} \\ &= -Q_B \mathbf{E}_{\mathrm{Q}M}(\mathbf{R}_B) + \sum_{A\in \mathrm{QM}} \nabla_B V^{\mathrm{LJ}}_{AB}\end{aligned} \tag{10.15}$$

for the derivatives with respect to the positions of the MM atoms B where $\mathbf{E}_{\mathrm{Q}M}$ is the electric field due to the QM charge distribution. The contributions to the gradient due to the point charge interactions and due to the interaction between the MM point charges and the QM electrons is evaluated by the QM program. Some QM programs do not calculate the forces acting on the MM atoms (point charges) due to the presence of the QM system but in general are able to return the electric field $\mathbf{E}_{\mathrm{Q}M}$ at arbitrary points in space which is then used to obtain these forces. The van der Waals repulsion (Lennard-Jones interaction) between QM and MM atoms is treated by AMBER in the same way as for semiempirical NDDO-type and DFTB methods.

10.2.2. General Remarks

When using the AMBER interface to external QM software packages for performing WFT or DFT based QM or QM/MM MD simulations, it is absolutely critical to be aware of the capabilities and limitations of the QM method to be employed. In particular, QM based MD can be more tricky than MM based MD in the sense that it is more likely that the QM program can fail for example due to SCF convergence problems. This can be the case if the geometry of the QM region is far from its ground state equilibrium, for example because a simulation is started from a bad geometry or performed at high temperature.

We have gone to large efforts and analyzed a large set of test simulations to provide the best default parameters for the supported QM programs such that forces are computed with sufficient accuracy to guarantee energy coservation for constant energy MD simulations. Of particular importance are SCF convergence and associated integral neglect thresholds and the size of the grid used for the numerical quadrature of the exchange-correlation (XC) potential and energy for DFT calculations. However, other than providing appropriate input parameters, AMBER does not have any control over the external program and it is at the user's discretion to employ sensible input parameters for the QM program and to prepare the system such that the simulations are started at a reasonable starting structure.

In any case we highly recommend to write restart files frequently so that a simulation can be restarted without loss of much computational time in the case that a simulation should crash. The interface also stores the last in- and output files of the external QM program during each MD step. Should there be any problems with the QM program, it is therefore possible to analyze the reasons and take appropriate countermeasures.

The interface requires data to be exchanged between *sander* and the QM program. The interface is based on file exchange and system calls and, during each step of a geometry optimization or an MD simulation, writes an input file for the external program, starts a single point gradient calculation with the external program, and reads the energy and forces from the external program's output file (binary ADF checkpoint or formatted GAMESS, Gaussian, ORCA, Q-Chem, MRCC or TeraChem output files). Data communication via MPI is also implemented and currently supported by TeraChem.

10.2.3. Limitations

In principle, all types of simulations that are possible with sander are supported. There are, however, some restrictions for simulations that require *sander* to run in parallel, in particular path integral molecular dynamics (PIMD) and replica exchange molecular dynamics (REMD), see the discussion of Parallelization below. The interface to external QM programs also lacks some features regarding solvent models in comparison to the semiempirical MNDO and DFTB QM/MM implementation that is available in AMBER, the most critical ones are listed here.

Generalized Born Generalized Born (GB) implicit solvent models are not supported if external QM programs are used for the QM region.

Particle Mesh Ewald (PME) and Periodic Boundary Conditions The PME approach for treating long-range electrostatic QM/MM and QM/QM interactions in periodic systems is currently not supported. It is possible to use periodic boundary conditions but a cutoff is used for the point charges to be included in the QM Hamiltonian (determined by *&qmmm* namelist variable *qmcut*) thus truncating the long-range QM/MM electrostatic interactions in (10.12). This leads to discontinuities in the potential energy surface and poor energy conservation for MD runs in the NVE ensemble. The user may consider running non-periodic simulations with a cutoff that is larger than the system size thus effectively including all interactions.

10.2.4. Performance Considerations

The computational cost of DFT is comparable to Hartree–Fock (HF) theory which is the simplest WFT method that serves as zeroth order approximation for more elaborate correlated WFT methods such as Møller–Plesset perturbation theory, configuration interaction theory and coupled cluster theory. The calculations can be accelerated by using density fitting approaches, sometimes called resolution-of identity (RI) approximation, which in the case of DFT with exchange-correlation (XC) functionals that do not require admixture of exact HF-exchange, leads to speedups of roughly one order of magnitude without compromising the accuracy of the results. Nevertheless, the computational cost of DFT is in general two to three orders of magnitude higher than that of semiempirical QM models. We recommend to carefully test the performance of the QM program to choose an optimal number of processor counts for parallelized QM calculations. Typical simulation performance for typical QM system sizes of tens of atoms will be on the order of a few picoseconds per day, depending on the underlying QM model chosen.

10.2.5. Parallelization

The MPI parallel executable *sander.MPI* can be used to run QM/MM MD simulations with external QM software in which the MM portion of the calculation is parallelized. However, the computational cost of the MM part is usually small compared to the cost of the QM part. In order to execute the QM part of the calculation in parallel, the external QM program has to be instructed to do so, as described in the sections below.

In the case of PIMD or REMD simulations that require a separate energy and force evaluation for each group at each time step, the parallelized executable *sander.MPI* has to be used. Multiple processes can be launched per group to parallelize the MM calculations. Care has to be taken to choose the right number of parallel threads in the external QM program. For example, on a machine with 32 cores, a simulation with 16 beads or replicas can run the external QM program with 2 threads in parallel to make maximum use of the available processing cores. If the available processors are spread over multiple nodes, special care has to be taken to ensure that the different instances of the external QM program are launched on the correct nodes.

It is possible to execute *sander.MPI* in parallel via MPI while also running MPI or OpenMP parallel versions of the external QM program. Depending on the MPI implementation, this can, however, fail. In our experience, MPICH and MVAPICH work well while we had problems with OpenMPI.

10.2.6. Usage

All that is required to use the interface is a working installation of AMBER and one or more of the supported QM programs. In order to use the external program from within *sander*, the *&cntrl* namelist variable *ifqnt = 1* must be set to enable QM calculations and the *&qmmm* namelist variable *qm_theory = 'EXTERN'* must be set to enable the external interface. The *&qmmm* namelist variable *qmmask* or *iqmatoms* is used for selecting the QM region just as for QM/MM calculations with the semiempirical NDDO-type and DFTB approaches that are natively available in AMBER. Charge and spin multiplicity for the QM region need to be defined via the variables *qmcharge* and *spin*, respectively, in the *&qmmm* namelist. For a QM MD simulation, the sander input file therefore needs to contain

```
! example input for QM simulation with external QM program
&cntrl
  ...
  ifqnt = 1,            ! switch on QM/MM
/
&qmmm
  qmmask = '@*',        ! select QM atoms (here: make all QM)
  qmcharge = 0,         ! charge on QM region (default = 0)
  spin = 1,             ! spin multiplicity of QM region (default = 1)
  qm_theory = 'EXTERN', ! use external QM program
/
```

For QM/MM simulations with electronic embedding (this is the default) we recommend to include all MM point charges as external electric field in the QM Hamiltonian to avoid problems with energy conservation. For non-periodic simulations this can be achieved by setting the *&qmmm* namelist variable *qmcut* to a value larger than the system size.

In addition either the *&adf*, *&gms*, *&nw*, *&gau*, *&orc*, *&qc, &mrcc* or *&tc* namelist must be present to use either ADF, GAMESS, NWChem, Gaussian, ORCA, Q-Chem or TeraChem, respectively, and to assign parameters for the external QM program. Please refer to the ADF, GAMESS, NWChem, Gaussian, ORCA, Q-Chem, MRCC or TeraChem user manual for details on settings for the *ab initio* or DFT calculations. A list of namelist variables and their default setting is given below. The defaults have been chosen such that energy conserving MD simulations in the NVE ensemble are possible. NWChem has not been extensively tested.

Properties that are calculated along the trajectory are printed to property files with names `adf_job.ext`, `gms_job.ext`, `gau_job.ext`, `orc_job.ext`, `qc_job.ext` and `tc_job.ext`, where `ext` is either `dip` for dipole moment (x, y, z component and absolute value) or `chg` for atomic charges, where supported. These property files are only written if requested and will be deleted at the beginning of a run, so back them up in case a trajectory needs to be restarted.

All calculations with a spin multiplicity larger than one will automatically be performed in the framework of an unrestricted formalism (as opposend to restricted open shell), that is with unrestricted HF (UHF), unrestricted DFT (UDFT) and MP2 with a UHF reference wave function (UMP2).

In addition to controlling the external programs via the *sander* input file, you may supply a template input file for the external program in order to provide additional input which is not supported by the external interface. The format, name, and input requirements for the template file vary with the external program as detailed in the corresponding program's documentation below. If you are using your own template, please make sure that the parameters of the QM method (like SCF convergence threshold and XC quadrature grid size) yield sufficiently accurate forces.

10.2.6.1. AMBER/ADF

To use ADF with the external interface, ADF must be properly installed on the working machine. In particular, the executable `adf` must be in the search path. By default the Becke integration grid with quality "goog" and the ZLM fit method with quality "good" is employed. If you prefer to use the old pair fit method (or are using an older ADF version that does not support the ZLM fit), we recommend to use "ZORA/QZ4P" basis set for the density fit for sufficiently accurate forces.

Limitations At present only mechanical embedding is supported.

&adf **Namelist variables**

basis	Basis set type to be used in the DFT calculation. Valid standard basis set types are: SZ, DZ, DZP, TZP, TZ2P, TZ2P+ and ZORA/QZ4P. (Default: basis = ’DZP’)
core	Type of frozen core to use. Allowed values are: None, Small, Medium, Large. (Default: core = ’None’)
zlmfit	Quality of density fit with the ZLM fit method. (Default: zlmfit = ’good’)
fit_type	Fit basis set type to be used for density fitting with the old pair fit method. Valid values are identical to the available basis sets (SZ, DZ etc) in which case the fit basis corresponding to the AO Basis will be used. By default the ZLM fit method will be used (Default: fit_type = ”)
xc	Exchange-correlation functional to be used. Popular choices are ’LDA VWN’, ’GGA BLYP’, ’GGA PBE’, ’HYBRID B3LYP’ and ’HYBRID PBE0’. Consult the ADF manual for all available options. (Default: xc = ’GGA BLYP’)
scf_iter	Maximum number of SCF cycles allowed. (Default: scf_iter = 50)
scf_conv	Threshold upon which to stop the SCF procedure. The tested error is the commutator of the Fock matrix and the density matrix. Convergence is considered to be achieved if the maximum element of the commutator (which is zero for an optimized wave function) is smaller than scf_conv. (Default: scf_conv = 1.0d-06)
beckegrid	Quality of Becke integration grid. Allowed values are: Normal, Good, VeryGood. (Default: core = ’Good’)
integration	Numerical integration accuracy for integration with olde teVelde-Baerends integration grid (Voronoi cells). By default the Becke grid will be used. The old integration grid can be used by specifying a number larger than 0, we recommend at least 5.0. (Default: integration = -1.0)
num_threads	Number of threads (and thus CPU cores) for ADF to use. Note that this is not requied if you are running in a queuing system as ADF will automatically use the full number of reserved cores. (Default: num_threads = 0 [this causes ADF to use all available cores on a machine])
use_dftb	Specifies whether DFTB shall be used with ADF’s DFTB program dftb. If use_dftb = 1 then DFTB will be used and only variables charge and scf_conv will be considered. (Default: use_dftb = 0 [do not use DFTB, regular DFT calculation]) - works only with older DFTB versions (prior to 2011).
exactdensity	The exact (as opposed to fitted) electron density is used for the evaluation of the exchange-correlation potential if exactdensity = 1. (Default: exactdensity = 0)
use_template	Determine whether or not to use a user-provided template file for running external programs. (Default: use_template = 0)
ntpr	Controls frequency of printing for dipole moment to file `adf_job.dip` (Defaults to &cntrl namelist variable ntpr)
dipole	Toggles writing of dipole moment to file `adf_job.dip` (Default: dipole = 0)

Example An input file for QM or (mechanical embedding) QM/MM MD with ADF using the PBE functional and the TZP basis set therefore would have to contain

```
&adf
  xc = 'GGA PBE',
  basis = 'TZP',
/
```

This would execute a simulation in which the Beckgrid with quality quality good and the ZLM fit with quality good are used (see default values above).

Template input file The template file for ADF should be named `adf_job.tpl` and must contain the following keywords:

```
BASIS ... END
SAVE TAPE21
```

You should not include the following (block) keywords in the template file as these are taken care of by *sander*:

```
UNITS
FRAGMENTS ... END
RESTART
GRADIENT
ATOMS ... END
```

10.2.6.2. AMBER/GAMESS-US

To use GAMESS with the external interface, GAMESS must be compiled on the target system. Make note of the version number you specify during the GAMESS compilation process (default is 00 which makes the GAMESS execution script `rungms` look for the executable `gamess.00.x`). If you use a different version number you must specify it with the *gms_version* namelist variable. `$GMS_PATH` should be set to the path where the script `rungms` is located (for example `/opt/gamess/`). We assume that the `rungms` script copies the output `.dat` files to the directory from which GAMESS is invoked. If this is not the case, please modify the script `rungms` accordingly.

Limitations Only mechanical embedding is supported with GAMESS. The available QM models are limited to HF, DFT and MP2 since only for these analytical gradients are available in GAMESS.

&gms Namelist variables

basis	Basis set type to be used in the calculation. Presently supported are the Pople type basis sets STO-3G, 6-31G, 6-31G*, 6-31G**, 6-31+G*, 6-31++G*, 6-311G, 6-311G* and 6-311G**. Also supported are the Karlsruhe valence triple zeta basis sets KTZV, KTZVP and KTZVPP (with none, one and two polarization functions, respectively) and the Dunning-type correlation consistent basis sets CCn (n = D, T, Q, 5, 6; officially called cc-pVnZ) and ACCn (as CCn but augmented with a set of diffuse function, officially called aug-cc-pVnZ). (Default: basis = "6-31G*")
method	QM method to be used. At present, we support 'HF' for Hartree–Fock, 'MP2' for second order Møller-Plesset perturbation theory and any of the supported DFT functionals. Popular choices for for DFT functionals include BP86, BLYP, PBE, B3LYP or PBE0. (Default: method = "BP86")
nrad	Number of radial points in the Euler-MacLaurin quadrature of the XC potential and energy density. (Default: nrad = 96)
nleb	Number of angular points in the Lebedev grids for the numerical quadrature of the XC potential and energy density. (Default: nleb = 590 [The GAMESS default of 302 is not accurate enough to conserve energy])

scf_conv	SCF convergence threshold. Convergence is reached when the absolute density change between two consecutive SCF cycles is less than scf_conv}. (Default: scf_conv = 1.0D-06)
maxit	Maximum number of SCF iterations. (Default: maxit = 50)
gms_version	This is the version number specified when building GAMESS. (Default: gms_version = 00)
num_threads	Number of threads (and thus CPU cores) for GAMESS to use. Note that GAMESS may require a special setup in the `rungms` script to be able to run using multiple threads. Unless num_threads is explicitly specified, GAMESS will only use one thread (run on one core). (Default: num_threads = 1)
mwords	The maximum replicated memory which your job can use, on every node. This is given in units of 1,000,000 words (as opposed to 1024*1024 words), where a word is defined as 64 bits. You may need to increase this value if GAMESS crashes due to not having enough memory allocated. (Default: mwords = 50)
use_template	Determine whether or not to use a user-provided template file for running external programs. (Default: use_template = 0)
ntpr	Controls frequency of printing for dipole moment and atomic charges to files `gms_prop.ext` (Defaults to &cntrl namelist variable ntpr)
chelpg	CHELPG charges are calculated if chelpg = 1. These charges are written to the file `gms_prop.chg` (Default: chelpg = 0)
dipole	Toggles writing of dipole moment to file `gms_prop.dip` (Default: dipole = 0)

Example An input file for QM or (mechanical embedding) QM/MM MD with GAMESS using the PBE functional and the 6-31G** basis set that should run GAMESS on 16 CPU cores therefore would have to contain

```
&gms
  method = 'DFT',
  dfttyp = 'PBE',
  basis  = '6-31G**',
  num_threads = 16,
/
```

Template input file The template file for GAMESS should be named `gms_job.tpl` and the `$CONTRL` card must contain the following keywords:

```
RUNTYP=GRADIENT
UNIT=ANGS
COORD=UNIQUE
```

You should not include the `$DATA` card in the template file as it is taken care of by *sander*.

10.2.6.3. AMBER/Gaussian

To use Gaussian with the interface, Gaussian03 or Gaussian09 must be properly installed on the system and the g03 or g09 executable must be in the path.

Limitations A cutoff is applied to QM/MM interactions in QM/MM simulations using electrostatic embedding with and without PBCs. This leads to discontinuities in the potential energy surface and poor energy conservation. In the case of QM/MM simulations without PBCs, this cutoff (*qmcut* variable in the *&qmmm* namelist) can be set to a number that is larger than the simulated system, thus effectively not applying a cutoff. This is recommended.

&gau Namelist variables

basis	Basis set type to be used in the calculation. Any basis set that is natively supported by Gaussian can be used. Examples are the single zeta, split valence or triple zeta Pople type basis sets STO-3G, 3-21G, 6-31G and 6-311G. The split-valence or triple zeta basis sets can be augmented with diffuse functions on heavy atoms or additionally hydrogen by adding one or two plus signs, respectively, as in 6-31++G. Polarization functions on heavy atoms or additionally hydrogens are used by adding one or two stars, respectively, as in 6-31G**. (Default: basis = "6-31G*")
method	Method to be used in the calculation. Can either be one of the WFT models for which Gaussian supports gradients, for example RHF or MP2, or some supported DFT functional. Popular choices are BLYP, PBE and B3LYP. (Default: method = "BLYP")
scf_conv	Threshold upon which to stop the SCF procedure. The tested error is the commutator of the Fock matrix and the density matrix. Convergence is considered to be achieved if the maximum element of the commutator (which is zero for an optimized wave function) is smaller than scf_conv}. Set in the form of 10^{-N}. (Default: scf_conv = 8)
num_threads	Number of threads (and thus CPU cores) for Gaussian to use. Unless num_threads is explicitly specified, Gaussian will only use one thread (run on one core). (Default: num_threads = 1)
use_template	Determine whether or not to use a user-provided template file for running external programs. (Default: use_template = 0)
ntpr	Controls frequency of printing for dipole moment to file `gau_job`.dip (Defaults to &cntrl namelist variable ntpr)
dipole	Toggles writing of dipole moment to file `gau_job`.dip (Default: dipole = 0)
mem	String that specifies how much memory Gaussian should be allowed to use. (Default: ’256MB’)

Example An input file for QM or QM/MM MD with Gaussian using the BP86 functional and the 6-31G** basis set and running in parallel on 8 threads (using 1 GB of memory) therefore would have to contain

```
&gau
  method = 'BP86',
  basis  = '6-31G**',
  num_threads = 8,
  mem='1GB',
/
```

Template input file The template file for Gaussian should be named `gau_job.tpl` and should only contain the route section of a Gaussian input file. The route section defines the method to be used and SCF convergence criteria. Charge and spin multiplicity are provided by via the *&qmmm* namelist. For example for a B3LYP calculation with 6-31G* basis set, the route section would be:

```
#P B3LYP/6-31G* SCF=(Conver=8)
```

Do not include any information about coordinates or point charge treatment since this will all be handled by *sander*. Also, do not include any *Link 0 Commands* (line starting with %) since these are handled by *sander*. If you want to run Gaussian in parallel, specify the number of processors via the *num_threads* variable in the *&gau* namelist.

10.2.6.4. AMBER/Orca

To use Orca with the interface, Orca must be properly installed on the system, the Orca executables need to reside in a directory that is in the search path. For convenience of use, namelist parameters in general correspond to Orca keywords, see the Orca manual for details.

Limitations A cutoff is applied to QM/MM interactions in QM/MM simulations with and without PBCs. This leads to discontinuities in the potential energy surface and poor energy conservation. In the case of QM/MM simulations without PBCs, this cutoff (*qmcut* variable in the *qmmm* namelist) can be set to a number that is larger than the simulated system, thus effectively not applying a cutoff. This is recommended.

Also note that ORCA only supports OpenMPI for parallel calculations.

&orc Namelist variables

basis	Basis set type to be used in the calculation. Possible choices include svp, 6-31g, etc. See Orca manual for a complete list. (Default: basis = "SV(P)")
cbasis	Auxillary basis set for correlation fitting. See Orca manual for a complete list. (Default: basis = "NONE")
jbasis	Auxillary basis set for Coulomb fitting. See Orca manual for a complete list. (Default: basis = "NONE")
method	Method to be used in the calculation. Popular choices include hf, pm3, blyp, and mp2. (Default: method = "blyp")
convkey	General SCF convergence setting for simplified Orca input. Can take values 'TIGHTSCF', 'VERYTIGHTSCF', etc. (Default: convkey='VERYTIGHTSCF')
scfconv	SCF convergence threshold for the energy. (Default: scfconv = -1, that is, not in use since we use the general convergence settings keyword *convkey*. Otherwise this would lead to SCF energy convergence of 10^{-N}au, if set to N.)
grid	Grid type used during the SCF for the XC quadrature in DFT. (Default: grid = 4, this corresponds to Intacc = 4.34 for the radial grid and an angular Lebedev grid with 302 points. Conservatively chosen together with finalgrid to conserve energy.)
finalgrid	Grid type used for the energy and gradient calculation after the SCF for the XC quadrature in DFT. (Default: finalgrid = 6, this corresponds to Intacc = 5.34 for the radial grid and an angular Lebedev grid with 590 points. Conservatively chosen together with *grid* to conserve energy.)
maxiter	Maximum number of SCF iteractions. (Default maxiter = 100)
maxcore	Global scratch memory (in MB) used by Orca. You may need to increase this when running larger jobs. See Orca manual for more information. (Default maxcore = 1024)
num_threads	Number of threads (and thus CPU cores) for Orca to use. Note that Orca only supports OpenMPI. (Default: num_threads = 1)
use_template	Determine whether or not to use a user-provided template file for running external programs. (Default: use_template = 0)
ntpr	Controls frequency of printing for the dipole moment to file `orc_job.dip` (Defaults to &cntrl namelist variable ntpr)
dipole	Toggles writing of the dipole moment to file `orc_job.dip` (Default: dipole = 0)

Example An input file for QM or QM/MM MD with Orca using the BLYP functional, the SVP basis set therefore would have to contain

```
&orc
  method = 'blyp',
  basis  = 'svp',
/
```

Template input file The template file for Orca should be named `orc_job.tpl` and must at least contain keywords specifying the method and basis set to be used in the calculation, for example:

```
# ORCA input file for BLYP/SVP simulation
! BLYP SVP
```

You should not include the following keywords in the template file as these are taken care of by sander (like setting the runtype and adding coordinates):

```
# NOT to be included in ORCA input file
!engrad
!energy  # (or any run type)
%pointcharges
*xyzfile # (or any coordinates)
```

10.2.6.5. AMBER/Q-Chem

To use Q-Chem with the interface, Q-Chem must be properly installed on the system. The q-chem executable needs to reside in a directory that is in the search path. For convenience of use, namelist parameters in general correspond to Q-chem keywords, see the Q-Chem manual for details. The interface has been tested with Q-Chem versions 4.0.0.1 and 4.1.1 for HF, DFT and MP2. Other methods have not been tested and could cause problems - please be careful and verify that forces/energies used by sander are correct in this case.

Limitations A cutoff is applied to QM/MM interactions in QM/MM simulations with and without PBCs. This leads to discontinuities in the potential energy surface and poor energy conservation. In the case of QM/MM simulations without PBCs, this cutoff (*qmcut* variable in the *qmmm* namelist) can be set to a number that is larger than the simulated system, thus effectively not applying a cutoff. This is recommended.

&qc Namelist variables

basis	Basis set type to be used in the calculation. Possible choices include ’6-31g**’, ’cc-pVDZ’ etc. See the Q-chem manual for a complete list. (Default: basis = ’6-31G*’ for DFT calculations and basis = ’cc-pVDZ’ for MP2)
auxbasis	Auxillary basis set for density fitting / RI methods. See Q-Chem manual for a complete list. (Default: basis = ’rimp2-cc-pVDZ’ for RI-MP2 calculations, otherwise none)
method	Method to be used in the calculation. Popular choices include ’BLYP’ or other density functionals, ’MP2’ and ’RIMP2’. Alternatively, the keywords exchange and correlation can be employed. (Default: method = ’BLYP’)
exchange	Exchange method. Can be specified together with the correlation keyword in place of the combined method keyword. (Default: exchange = ”)
correlation	Correlation method. Can be specified together with the exchange keyword in place of the combined method keyword. (Default: correlation = ”)
scf_conv	SCF convergence threshold. (Default: scfconv = 6)
num_mpi_prcs	Number of MPI processes for Q-Chem to use. The total number of CPUs to be used is num_mpi_prcs times num_threads. (Default: num_mpi_prcs = 1)
num_threads	Number of threads for Q-Chem to use for each MPI process. Really this is number of threads. The total number of CPUs to be used is num_mpi_prcs times num_threads. (Default: num_threads = 1)
use_template	Determine whether or not to use a user-provided template file for running external programs. (Default: use_template = 0)

ntpr Controls frequency of printing for the dipole moment to file `qc_job.dip` (Defaults to &cntrl namelist variable ntpr)

dipole	Toggles writing of the dipole moment to file `qc_job.dip`. This is currently not supported. (Default: dipole = 0)
guess	Toggles use of MOs from previous step as initial guess to accelerate SCF convergence. Any string different from 'read' will disable this. (Default: guess = 'read')

Example An input file for QM or QM/MM MD with Q-Chem using MP2 with the cc-pVTZ basis set therefore would have to contain

```
&qc
  method = 'mp2',
  basis  = 'cc-pVTZ',
/
```

Template input file The template file for Q-chem must be named `qc_job.tpl` and must only contain keywords in the Q-Chem $rem input section that specify the QM method and basis set to be used in the calculation, for example:

```
EXCHANGE becke
CORRELATION lyp
BASIS 6-311G**
SCF_CONVERGENCE 7
```

The interface will take care of adding other keywords to the $rem section such as JOBTYPE and writing the $molecule input file sections.

10.2.6.6. AMBER/MRCC

To use MRCC with the interface, the MRCC program suite must be properly installed on the system. The MRCC driver program dmrcc needs to reside in a directory that is in the search path. For convenience of use, namelist parameters in general correspond to MRCC keywords, see the MRCC manual for details. The interface has been tested with the MRCC release from July 15, 2016 for HF and DFT. Other methods have not been tested but should also work - please be careful and verify that forces/energies used by sander are correct in this case.

Limitations A cutoff is applied to QM/MM interactions in QM/MM simulations with and without PBCs. This leads to discontinuities in the potential energy surface and poor energy conservation. In the case of QM/MM simulations without PBCs, this cutoff (*qmcut* variable in the *qmmm* namelist) can be set to a number that is larger than the simulated system, thus effectively not applying a cutoff. This is recommended.

&mrcc Namelist variables

basis	Basis set type to be used in the calculation. Possible choices include '6-31g**', 'cc-pVDZ' etc. See the MRCC manual for a complete list. (Default: basis = '6-31G*')
calc	Type of calculation, e.g. 'SCF', 'B3LYP', 'MP2', 'CCSD(T)', etc. (Default: calc = 'SCF')
dft	Can be specified to request a DFT calculation and specify the DFT method. (Default: dft = 'off')
mem	Memory that will be allocated for the calculation. (Default: mem = '256MB')
verbosity	Controls the verbosity of the MRCC output file. (Default: verbosity = 2)

ntpr	Controls frequency of printing for the dipole moment to file `mrcc_job.dip` (Defaults to &cntrl namelist variable ntpr)
do_dipole	Toggles writing of the dipole moment to file `mrcc_job.dip`. (Default: dipole = 0)
nprintlog	Frequency of storing MRCC output files during a minimization of molecular dynamics run. (Default: keep only last output file, nprintlog = 0)
debug	Toggles debug mode, which prints subroutine calls and additional information about the AMBER/MRCC interface. (Default: no debugging, debug = 0)
use_template	Requests use of a template file to generate MRCC input files to utilize all the capabilities of that are not available through &mrcc namelist keywords. The template file is basically a truncated MINP file (the default input file for MRCC) which only includes the MRCC keywords. (Default: do not use a template input file, use_template = 0)

The following &mrcc namelist variables control multilayer calculations (i.e. QM/QM/MM or QM/QM/QM/MM embedding[302]; the region highlighted in bold is controlled by the keyword). Only single point calculations are currently possible with such mulitlayer calculations.

embed	Specifies the method of the embedding QM region (2. layer) in a QM/**QM**/MM (3 layer) calculation or specifies the method of the 3. layer in a QM/QM/**QM**/MM (4 layer) calculation. Please read the MRCC manual for available options. (Default: embed = ’off’)
embedatoms	Specifies the active atoms of the embedded QM region (1. layer) in a **QM**/QM/MM (3 layer) calculation or specifies the active atoms of the 1. and 2. layer in a **QM**/**QM**/QM/MM (4 layer) calculation. Comma separated list of integers (Default: embedatoms = 0)
nmo_embed	Specifies the number of active MOs of the embedded QM region (1. layer) in a **QM**/QM/MM (3 layer) calculation or specifies the number of active MOs of the 1. and 2. layer in a **QM**/**QM**/QM/MM (4 layer) calculation. **= 0** The program automatically determines the MOs of the active region with the Boughton-Pulay (BP) algorithm. (default) **> 0** Number of MOs that will be selected based on the Mulliken charges of the active atoms.
corembed	Specifies the low-level correlation method of the embedding QM region (2. layer) in a QM/**QM**/MM (3 layer) calculation or specifies the low-level correlation method of the 2. layer in a QM/**QM**/QM/MM (4 layer) calculation. Please read the MRCC manual for available options. (Default: corembed = ’off’)
corembedatoms	Specifies the active atoms of the embedded QM region (1. layer) in a **QM**/QM/MM (3 layer) calculation or specifies the active atoms of the 1. layer in a **QM**/QM/QM/MM (4 layer) calculation. Please note that the corembedatoms have to be a subset of the embedatoms if a 4 layer calculation is requested. Comma separated list of integers (Default: corembedatoms = 0)
nmo_corembed	Specifies the number of active MOs of the embedded QM region (1. layer) in a **QM**/QM/MM (3 layer) calculation or specifies the number of active MOs of the 1. layer in a **QM**/QM/QM/MM (4 layer) calculation. **= 0** The program automatically determines the MOs of the active region with the Boughton-Pulay (BP) algorithm. (default) **> 0** Number of MOs that will be selected based on the Mulliken charges of the active atoms.

Examples An input file for QM or QM/MM MD with MRCC using DFT with the BLYP functional and the cc-pVTZ basis set therefore would have to contain

```
&mrcc
  calc = 'blyp',
  basis  = 'cc-pVTZ',
/
```

An example input for a multilayer QM/QM/MM calculation with LCCSD(T) for a subset of QM atoms 7 to 12 embedded into the remainder of the QM region described by PBE (i.e. LCCSD(T)/PBE/MM) would be

```
&mrcc
  calc = 'LCCSD(T)',
  basis  = 'cc-pVTZ',
  embed = 'PBE',
  embedatoms = 7,8,9,10,11,12
/
```

This assumes that atoms 7-12 are part of the QM region. A 4-layer QM/QM/QM/MM calculation with LCCSD(T) for atoms 7 to 12 embedded into LMP2 for atoms 13 to 16 and the remainder described by PBE (i.e. LCCSD(T)/LMP2/DFT/MM) would be requested with

```
&mrcc
  calc = 'LCCSD(T)',
  basis  = 'cc-pVTZ',
  embed = 'PBE',
  embedatoms = 7,8,9,10,11,12,13,14,15,16,
  corembed = 'LMP2',
  corembedatoms = 7,8,9,10,11,12,
/
```

Template input file The template file for MRCC must be named `mrcc_job.tpl` and must only contain keywords that specify the QM method and basis set to be used in the calculation. Not to be included are following keywords: qmmm, qmreg, dens, pointcharges, geom, embed, corembed, scfiguess. The interface will take care of adding other keywords and writing the coordinate input file section.

10.2.6.7. AMBER/TeraChem

To use TeraChem with the interface, TeraChem must be properly installed on the system. In particular, the `terachem` executable needs to be in the search path. Namelist parameters correspond to TeraChem keywords, see the TeraChem manual for details.

Limitations A cutoff is applied to QM/MM interactions in QM/MM simulations with and without PBCs. This leads to discontinuities in the potential energy surface and poor energy conservation. In the case of QM/MM simulations without PBCs, this cutoff (*qmcut* variable in the *&qmmm* namelist) can be set to a number that is larger than the simulated system, thus effectively not applying a cutoff. This is recommended.

&tc Namelist variables

basis	Basis set type to be used in the calculation. Possible choices presently (TeraChem version 1.4) are ’STO-3G’, ’3-21G’, ’6-31G’ and ’6-311G’, ’3-21++G’ and ’6-31++G’ (Default: basis = ’6-31G’)

method	Method to be used in the calculation, can be either ’RHF’ or some supported DFT functional. Popular choices are ’BLYP’, ’PBE’ and ’B3LYP’. (Default: method = ’BLYP’)
dftd	Determines whether dispersion corrections are applied in the case of DFT calculations. (Default: dftd = ’no’)
precision	Precision model setting (single vs double precision). (Default: precision = ’mixed’)
dynamicgrid	Use coarse grid during early SCF iterations. (Default: dynamicgrid = ’yes’)
threall	Determines a variety of thresholds. (Default: threall = 1.0E-11)
convthre	SCF convergence threshold for the wavefunction. (Default: convthre = 3.0E-05, which leads to SCF energy convergence of approximately 10^{-7} au or 10^{-4} kcal/mol)
maxit	Maximum number of SCF iterations. (Default: maxit = 100)
dftgrid	DFT grid to be employed for the numerical XC quadrature in DFT calculations. (Default: dftgrid = 1)
ngpus	Determines how many GPUs are to be used. (Default: ngpus = 0, which uses all available GPUs)
gpuids	If *ngpus* has a value other than zero, this determines the IDs of the GPUs to be used for the calculation. (Default: gpuids = 0, 1, 2, etc.)
executable	Name of the TeraChem executable. (Default: executable = terachem)
use_template	Determine whether or not to use a user-provided template file for running external programs. (Default: use_template = 0)
ntpr	Controls frequency of printing for dipole moment and atomic charges to files `tc_job.ext`. (Defaults to &cntrl namelist variable ntpr)
charge_analysis	Toggles writing of atomic charges to file `tc_job.chg` (Options: ’none’ or ’Mulliken’. Default: dipole = ’none’)
dipole	Toggles writing of dipole moment to file `tc_job.dip` (Default: dipole = 0)

Example An input file for QM or QM/MM MD with TeraChem using the PBE functional and the 6-31G* basis set therefore would have to contain

```
&tc
  method = 'PBE',
  basis  = '6-31G*',
/
```

Template input file The template file for Terachem should be named `tc_job.tpl` and must at least contain the following keywords:

```
basis
method
```

Any content of the template file after a line containing the `end` keyword will be ignored.

You should not include the following keywords in the template file as these are taken care of by *sander*. Instead, specify these via the *&qmmm* or *&tc* namelist:

```
run
charge
spinmult
coordinates
pointcharges
amber
gpus
```

10.3. Adaptive solvent QM/MM simulations

Traditional QM/MM approaches are based on a static QM/MM partitioning in which atoms belonging to the QM and MM regions are selected at the beginning of a molecular dynamics simulation. Such a static partitioning cannot be applied if part of the bulk solvent in the vicinity of a region of interest needs to be included in the QM region. Examples include cases in which the bulk solvent participates directly in a chemical reaction or in which important interactions between the solute and the bulk solvent, such as polarization and charge transfer, are not well parameterized at the QM/MM level and thus need to be described quantum mechanically. Due to molecular diffusion, solvent molecules will constantly exchange between the QM and MM regions and thus require a special treatment.

Several approaches have been developed that allow molecules to change their QM or MM character when crossing the boundaries between the QM and MM regions. A good overview and comparison of these approaches is available in the work by Bulo *et al.*[303]. One of the most accurate approaches is the difference-based adaptive solvation (DAS) method[304], in the following simply called adaptive QM/MM (adQM/MM). This method is available in Amber through a parallelized implementation that has been developed by Andreas Goetz (SDSC) with help from Ross Walker (SDSC), Rosa Bulo (Utrecht University) and Kyoyeon Park (UCSD). The usefulness of this adQM/MM approach for aqueous systems has been demonstrated with a development version of this implementation[305]. If you publish work that results from using this implementation, please cite the following work:

- A. W. Götz, K. Park, R. E. Bulo, F. Paesani, R. C. Walker, *Efficient adaptive QM/MM implementation: Application to ion binding by peptides in solution*, in preparation.
- R. E. Bulo, B. Ensing, J. Sikkema, L. Visscher, *Toward a practical method for adaptive QM/MM simulations*, J. Chem. Theory Comput. **9**, 2212-2221 (2009), DOI: 10.1021/ct900148e

In what follows we will describe the theoretical background of this implementation and how to perform adQM/MM simulations. For an alternative approach, see section 10.4.

10.3.1. Theoretical background

In adQM/MM simulations, we distinguish three different regions, an active region (A), a transition region (T), and the environment region (E). The active region contains both the part of the system that is permanently treated quantum mechanically (similar to the QM region in regular QM/MM simulations) and the solvent molecules in its vicinity that are also treated quantum mechanically. The E region is the part of the system that is treated at the MM level. Within the T region, molecules change their character from purely QM to purely MM, that is, molecules in the T region have partial QM and MM character, depending on their position within the T region. The T region that connects the A and E regions is required to guarantee that the potential energy surface or forces remain continuous throughout the simulation.

10.3.1.1. System partitioning

In the adQM/MM method[304], a partial MM character λ is assigned to each molecule in the T region. The value of λ depends on the distance of the molecule from the center of the A region according to

$$\lambda(r) = \begin{cases} 0 & r \leq R_\mathrm{A} \\ \frac{(r-R_\mathrm{A})^2(3R_\mathrm{T}-R_\mathrm{A}-2r)}{(R_\mathrm{T}-R_\mathrm{A})^3} & R_\mathrm{A} < r < R_\mathrm{T} \\ 1 & r \geq R_\mathrm{T} \end{cases}, \tag{10.16}$$

where R_A and R_T are the inner and outer radii delimiting the T region. The switching function thus interpolates smoothly between QM (A region) and MM (E region).

The adQM/MM energy can be constructed as a weighted average of regular QM/MM energies due to all possible 2^{N_T} partitionings in which the N_T molecules in the T region are assigned either to the QM or the MM region,

$$E^{\mathrm{ad}QM/MM} = \sum_a \sigma_a E_a^{\mathrm{QM}/MM}. \tag{10.17}$$

The statistical coefficients σ_a for the QM/MM partitionings are defined on basis of the λ values defined above,

$$\sigma_a = \begin{cases} 0 & if\, \max(\{\lambda\}_a^{QM}) > \min(\{\lambda\}_a^{\mathrm{MM}}) \\ \min(\{\lambda\}_a^{\mathrm{MM}}) - \max(\{\lambda\}_a^{\mathrm{QM}}) & if\, \max(\{\lambda\}_a^{\mathrm{QM}}) \leq \min(\{\lambda\}_a^{\mathrm{MM}}) \end{cases}, \tag{10.18}$$

where $\{\lambda\}_a^{QM}$ and $\{\lambda\}_a^{\mathrm{MM}}$ are the sets of λ values for a given QM/MM partitioning a that are assigned to the QM and MM regions, respectively. Due to this choice of coefficients, the weight σ_a of a QM/MM partitioning is zero if the partition contains one or more MM molecules closer to the A region than any of the QM molecules. The total number of nonzero QM/MM partitionings in an adQM/MM simulations is thus $N_\mathrm{T}+1$. In addition it is guaranteed that the weight of each partition varies smoothly from 0 to 1, removing discontinuities in the system dynamics that would appear in standard QM/MM simulations if a molecule would change its character by diffusing in or out of the QM region.

10.3.1.2. Force interpolation

The forces resulting from the adQM/MM energy 10.17 are a weighted sum of the force from each nonzero QM/MM partitioning and also contain a term that depends on the derivatives of the weight functions,

$$\mathbf{F}^{\mathrm{ad}QM/MM} = -\sum_a \left[\sigma_a \frac{\partial E_a^{\mathrm{QM}/MM}}{\partial \mathbf{R}} + \frac{\partial \sigma_a}{\partial \mathbf{R}} E_a^{\mathrm{QM}/MM}\right]. \tag{10.19}$$

This introduces an artificial dependence on the relative energies of the different QM/MM partitionings. Thus, in place of the energy interpolation scheme, a force interpolation is applied in which the forces are given as

$$\tilde{\mathbf{F}}^{\mathrm{ad}QM/MM} = -\sum_a \sigma_a \frac{\partial E_a^{\mathrm{QM}/MM}}{\partial \mathbf{R}}. \tag{10.20}$$

The force interpolation does not conserve the energy from equation 10.17 but it is possible to define a conserved quantity according to

$$\tilde{E}^{\mathrm{ad}QM/MM} = E^{\mathrm{ad}QM/MM} + W, \tag{10.21}$$

where the correction term W is defined through

$$\frac{\partial W}{\partial \mathbf{R}} = -\sum_a \frac{\partial \sigma_a}{\partial \mathbf{R}} E_a^{\mathrm{QM}/MM}. \tag{10.22}$$

The quantity $\tilde{E}^{\mathrm{ad}QM/MM}$ is not a potential energy since it is only defined along the path taken by the system during the simulation. It is nevertheless useful to monitor this quantity to determine whether the simulation settings lead to numerical stability. The correction term W can be expressed as the path integral of its force vector from equation 10.22, which can be discretized. For step n of an MD simulation it is given as

$$W_n = \sum_i^n \sum_a E_a^{QM/MM}(i) \frac{\sigma_a(i+1) - \sigma_a(i-1)}{2}. \tag{10.23}$$

The Amber implementation uses exclusively the force interpolation scheme from equation 10.20 and optionally computes the correction term W from equation 10.23 to enable monitoring of the conserved quantity $\tilde{E}^{adQM/MM}$ from equation 10.21.

10.3.1.3. Alternative definitions of active, transition and environment regions

So far we have defined the boundaries between the A, T, and E regions with the distances R_A and R_T from the center of the active region. In this case both the A and the T regions have fixed volumes but the number of solvent molecules inside each region can vary during the simulation. Alternatively, we can fix the number of solvent molecules N_A and N_T within the A and T regions, respectively. In this case the volume of the A and T regions as well as the radii R_A and R_T will vary during the course of a simulation. The advantage of fixing the number of solvent molecules in the T region is that the number of QM/MM partitionings that needs to be considered also remains constant ($N_\mathrm{T}+1$). This is useful to optimize load balancing in a parallel adQM/MM implementation. The downside is that expression 10.23 does not strictly hold any more since the coefficients σ_a depend on the λ values which in turn depend on R_A and R_T. However, in practice, this is usually not an issue since the conserved quantity $\tilde{E}^{adQM/MM}$ needs monitoring only during simulation setup to choose settings that afford sufficient numerical stability. One thus can test simulation settings with fixed radii R_A and R_T and then switch to fixed molecule numbers N_A and N_T for production runs.

10.3.2. Running adQM/MM simulations with sander

Performing simulations with the adQM/MM approach described above requires the MPI parallelized *sander* executable `sander.MPI`. The implementation features a dual layer parallelization in which the calculations for all individual QM/MM partitionings are performed in parallel. Each of these QM/MM calculations can in turn be run in parallel. The parallelization across QM/MM partitionings is based on the *multisander* code infrastructure which effectively runs independent copies of *sander* for each QM/MM partitioning (similar to the replica exchange, path integral and thermodynamic integration implementations).

In order to run an adQM/MM simulation, the `mdin` input file needs to be set up similar to a regular QM/MM simulation. The QM region as defined in the *&qmmm* namelist defines the atoms that are in the permanent QM region. In addition, the *&qmmm* namelist variable *vsolv* needs to be set to 2 or 3 for fixed number of molecules in the A and T region or fixed size of A and T region, respectively. The following shows the minimum additions to the `mdin` input file that are required to perform an adQM/MM simulation as compared to a traditional QM/MM simulation with fixed QM and MM regions:

```
# mdin file - minimum additional content for adaptive solvent QM/MM
&qmmm
  ...
  adjust_q = 0, ! required, charge cannot be redistributed
  vsolv = 2,    ! switch on adQM/MM with fixed molecule numbers
                !                         in A and T region
/
&vsolv
  nearest_qm_solvent = 6, ! number of solvent molecules in A region
/
&adqmmm
  n_partition = 4, ! number of QM/MM partitionings
                   ! = number of molecules in T region + 1
/
```

In this example, a fixed number of solvent molecules is contained in the A region (6) and in the T region (3, since the number of QM/MM partitionings is $N_\mathrm{T}+1$). Thus, the volume of the A and T regions changes during the simulation. Details of all namelist variables are collected below.

In addition, a groupfile for *multisander* is required. This groupfile should point all *sander* copies to the same `mdin` input file, `inpcrd` coordinate file and `prmtop` parameter and topology file:

```
# groupfile for adaptive solvent QM/MM run with n_partition = 4
-O -i mdin -c inpcrd -p prmtop
-O -i mdin -c inpcrd -p prmtop
-O -i mdin -c inpcrd -p prmtop
-O -i mdin -c inpcrd -p prmtop
```

If you explicitly specify output file names, make sure to give separate names to each group (for example `mdout.000`, `mdout.001` etc), see also the *multisander* documentation. *The mutisander* adQM/MM simulation can then be executed with

```
mpirun -np 4 sander.MPI -rem 0 -ng 4 -groupfile groupfile
```

In this example, 4 MPI processes will be launched for 4 process groups (*sander* copies). The individual QM/MM calculations for each partitioning would thus run in serial. To run the individual QM/MM calculations in parallel, the number of MPI processes must be a multiple of the number of process groups.

Adaptive solvent QM/MM simulations can be performed both with the semiempirical NDDO-type and DFTB methods that are native to *sander* or with QM methods that are available via the interface to external QM programs. In the latter case, each process group will launch only one instance of the external QM program and the parallelization of the QM part of the QM/MM calculations is determined by the settings for the external QM program.

10.3.2.1. Important notes for system preparation and adQM/MM simulations

At the time of writing (release of AMBER 16) there is only a limited body of experience with adQM/MM simulations documented in the literature. Running adQM/MM simulations requires careful simulation setup, in particular regarding the size of the A and T regions. The A region needs to be sufficiently large to correctly describe the physics of the system of interest. The T region on the other hand needs to be sufficiently large to minimize force interpolation errors between the QM and MM regions. Since the cost of an adQM/MM simulation scales linearly with the number of molecules in the T region, a tradeoff between accuracy and cost often needs to be made. This in turn might lead to simulations that behave nicely for many time steps but eventually experience sudden, large (unphysical) forces on atoms at the T region boundaries. Similarly, whether it is more appropriate to define the center of the A region via an atom or the center of mass of the permanent QM region will affect the numerical stability of a simulation, depending on the particular system. Likewise for determining the distances of the solvent molecules via an atom or the center of mass of the solvent. In the case of water as solvent, problems can arise due to autoprotolysis which can lead to the formation of hydroxide and hydronium ions in the A region. Since the MM force field is not parameterized for hydroxide or hydronium ions, these will experience strong (unphysical) forces upon entering the T region. Careful monitoring of adQM/MM simulations and a bit of patience is thus advisable. It is a good idea to monitor the size of the A and T region and to check coordinates of atoms in the QM regions of all partitionings.

10.3.2.2. Namelist parameters for adaptive solvent QM/MM simulations

Adaptive solvent QM/MM simulations require setting the *vsolv* variable in the *&qmmm* namelist and setting variables in the *&vsolv* and *&adqmmm* namelists.

&vsolv namelist parameters The *&vsolv* namelist contains parameters that describe which solvent molecules are contained in the A region in addition to the permanent QM region that is defined in the *&qmmm* namelist. This namelist can be used without the *&adqmmm* namelist in a regular QM/MM simulation with *sander* if the variable *vsolv* in the *&qmmm* namelist is set to 1 instead of 2 or 3 (see 10.1.6). In this case there is no transition region and solvent molecules entering / leaving the QM region during the simulation would switch abruptly between QM and MM description. This is not recommended since it will results in large unphysical forces whenever such a switch occurs. However, this option is useful for post-processing of trajectories with single point QM/MM calculations in which the solvent molecules closest to the permanent QM region are treated quantum mechanically.

nearest_qm_solvent_resname Residue name of the solvent that can exchange between QM and MM region (Default: nearest_qm_solvent_resname = 'WAT')

nearest_qm_solvent Number of solvent molecules in the A region (Default: nearest_qm_solvent = 0)

nearest_qm_solvent_fq Frequency of updating of the A region. Should be set to 1 (every MD step) for adQM/MM simulations. (Default: nearest_qm_solvent_fq = 1)

nearest_qm_solvent_center_id Determines the atom(s) of the solvent molecules that is used to calculate the distance to the QM region.

- **= 0** Use the atom that is closest to the QM region. (default)
- **= -1** Use the center of mass.
- **> 0** Use this atom number within the solvent residue.

qm_center_atom_id Determines the atom of the permanent QM region that is used to calculate the distance to the solvent molecules.

- **= 0** Use the atom of the permanent QM region that is closest to a solvent molecule. Not supported for adQM/MM since the radii of the A and T region would remain undefined - a common point of reference is required for all solvent molecules. Useful only for post-processing of trajectories. (default)
- **= -1** Use the center of mass of the permanent QM region.
- **> 0** Use this atom number. Note that this is an absolute atom number - obviously, you should choose an atom that is in the permanent QM region.

verbosity Controls verbosity of vsolv output in the *mdout* file.

- **= 0** Standard verbosity. (default)
- **> 1** Increase verbosity.

&adqmmm namelist parameters If the *&qmmm* namelist variable *vsolv* is set to 2 or 3, an adQM/MM simulation with a fixed number of molecules in the A and T regions or fixed size of the A and T regions, respectively, is requested. Details of the adQM/MM simulation are set in the *&adqmmm* namelist as follows.

n_partition Defines the number of QM/MM partitions to be used. For vsolv=2 this also determines the number of solvent molecules in the transition region, which is n_partition - 1. For vsolv=3 it has to be set to the largest number of QM/MM partitionings that will be encountered for the chosen values of RA and RT. (Default: n_partition = 1)

RA Defines the radius R_A of the A region in Angstrom. Only relevant for vsolv=3. Needs to be changed from the default value and requires setting of RT. (Default: RA = -1.0)

RT Defines the radius R_T of the T region in Angstrom. Only relevant for vsolv=3. Needs to be changed from the default value and requires setting of RA. (Default: RT = -1.0)

calc_wbk Controls whether the book-keeping term W is calculated.

- **= 0** Do not calculate W. (default)
- **= 1** Calculate W via one-sided difference approximation (not recommended).
- **= 2** Calculate W via central-difference approximation, see equation 10.23. Requires additional computations for (dis)appearing partitionings. (recommended if W is desired).

verbosity Controls verbosity of adQM/MM output in the *mdout* file.

- **= 0** Standard verbosity. (default)
- **= 1** Increase verbosity - write distances of residues in T region from center of A region to file `adqmmm_res_distances.dat` and σ_a values to file `adqmmm_weights.dat`. These files get overwritten at each program start.
- **= 2** Increase verbosity - write distances and σ_a values also to the *mdout* file. Also write λ values.

print_qm_coords Controls whether coordinates of the QM atoms in each partitioning are written to file.

- **= 0** Do not write coordinates. (default)
- **= 1** Write QM coordinates for all QM/MM paritionings in xyz format to files `QM_coords.001` etc. Files are overwritten upon each program call.

10.4. Adaptive buffered force-mixing QM/MM

10.4.1. Introduction

In hybrid quantum mechanical – molecular mechanical (QM/MM) methods the reactive part of the system (i.e. where a significant change of the charge density distribution is expected) is described using a quantum mechanical model while the rest of the system is treated using molecular mechanics. Conventional (“energy-mixing”) QM/MM methods (convQM/MM) define a unique total energy function for the whole system that consists of three terms: the energy of the QM model applied to the atoms in the QM region, the energy MM model applied to atoms in the MM region and the interaction energy between the two regions:

$$E^{\mathrm{QM/MM}}(\mathrm{QM+MM}) = E^{\mathrm{QM}}(\mathrm{QM}) + E^{\mathrm{MM}}(\mathrm{MM}) + E^{\mathrm{QM}\leftrightarrow\mathrm{MM}}(\mathrm{QM+MM}), \tag{10.24}$$

where the superscript represents the level of theory, while the region to which they are applied are indicated in parentheses. The coupling between the quantum region and the surrounding atoms ($E^{\mathrm{QM}\leftrightarrow\mathrm{MM}}(\mathrm{QM+MM})$) can be taken into account in several ways. For example, in the more sophisticated approaches, the effects of the MM charges are included in the quantum mechanical SCF calculation in the form of an externally applied field. Given a total energy, performing Hamiltonian or any other standard dynamics is straightforward. However, several uncontrolled errors could potentially be introduced by such schemes. Representing the environment by a set of point charges can over-polarise the QM region, and conversely the electrostatic effect of the ever-changing quantum mechanical charge density on atoms at the edge of the MM region is quite different from what is assumed when the MM force field parameters are determined. The delicate balance that exists between the various non-bonded MM terms is therefore no longer maintained across the QM-MM boundary. Furthermore, if adaptivity, i.e. transitions of atoms between the two regions, is allowed, a new problem appears: in general there can be chemical potential differences between the QM and MM regions for various species, and this results in a net flow betwen the regions, leading to unphysical density differences, structure and dynamics. Allowing adaptivity in this sense can be important when the active region itself is mobile (e.g. penetration, adhesion, crack propagation), or diffusional processes in the environment are relevant (e.g. water molecules, ions, residues enter and exit the QM region during the dynamics). To overcome these problems the adaptive buffered “force-mixing” QM/MM (abfQM/MM) method was introduced [306, 307]. The implementation of abfQM/MM was carried out by Letif Mones (University of Cambridge) and Gabor Csanyi (University of Cambridge) with help from many others (see the article below). When using this implementation in your work please cite the following papers:

- Noam Bernstein, Csilla Várnai, Iván Solt, Steven A. Winfield, Mike C. Payne, István Simon, Mónika Fuxreiter and Gábor Csányi, *QM/MM simulation of liquid water with an adaptive quantum region,* Phys. Chem. Chem. Phys., **14**, 646–656 (2012), DOI: 10.1039/c1cp22600b
- Csilla Várnai, Noam Bernstein, Letif Mones and Gábor Csányi, *Tests of an Adaptive QM/MM Calculation on Free Energy Profiles of Chemical Reactions in Solution,* |J. Phys. Chem. B, **117**, 12202–12211 (2013), DOI: 10.1021/jp405974b

- Letif Mones, Andrew Jones, Andreas W. Götz, Teodoro Laino, Ross C. Walker, Ben Leimkuhler, Gábor Csányi and Noam Bernstein, *Implementation of the Adaptive Buffered Force QM/MM method into CP2K and Amber program packages*, in preparation.

10.4.2. Technical details of abfQM/MM

In the abfQM/MM method two independent force calculations are performed at each MD step. The first and more time consuming calculation is an extended conventional QM/MM calculation, which is used for calculating the forces of atoms treated quantum mechanically during the dynamics. We start with a *core* QM region, which comprises atoms that will always be treated quantum mechanically throughout the simulation. This region is enlarged (using a distance criterion, see below) to obtain the *dynamical* QM region which contains the atoms that follow QM forces. The dynamical QM region is surrounded by a buffer region whose size can be determined by simple force convergence tests [308, 309] and its construction in practice is based on geometrical considerations: atoms or molecules that are within a specified distance from the dynamical QM region are added to the buffer region. From this first calculation only the forces of the atoms in the dynamical QM region are kept and the rest (namely the forces on atoms in the buffer region) are discarded. The second calculation is used for obtaining good forces on MM atoms, especially important near the QM/MM boundary. For this, either fully MM representation of the whole system is used or, alternatively, another QM/MM force calculation, but this time using a smaller (*reduced*) QM region consisting of only the atoms in the *core* QM region. The abfQM/MM method is an abrupt force mixing method, which means that the forces are not derived from a total energy expression but a simple combination of the forces of the two calculations described above

$$\mathbf{F}_i^{\text{abfQM/MM}}(\text{QM+MM}) = \begin{cases} \mathbf{F}_i^{\text{Extended}} & \text{if } i \text{ is in the dynamical QM region,} \\ \mathbf{F}_i^{\text{Reduced}} & \text{otherwise,} \end{cases} \tag{10.25}$$

where the superscripts Extended and Reduced denote that the forces are taken from the first and second calculations described above, respectively. The selection of the QM and buffer atoms is controlled by distance criteria. Using a single distance criterion measured from some key atoms in the QM region, however, would lead to rapid fluctuation in the region definitions because atoms may cross and re-cross repeatedly. To reduce this effect, a hysteretic algorithm can be applied using an inner (r_{in}) and an outer (r_{out}) radius [306]. Thus, an MM atom is redesignated to be QM if its distance measured from the QM region (as defined by a set of atoms *always* treated quantum mechanically) is less than r_{in} and a QM atom is redesignated to be MM if this distance is larger than r_{out}. Similar hysteretic algorithms are applied for the definition of the dynamical QM region as well as the buffer region.

The above definitions may lead to QM atoms that have covalent chemical bonds with MM atoms. This is not necessarily a problem, as these bonded interactions can be treated in several ways from the point of view of carrying out the the QM/MM calculation (e.g. link atoms, special pseudopotentials, frozen localized orbitals etc.). However, none of these schemes are general, i.e. cutting some type of QM-MM bonds in this way might not yield reasonable forces. For example, highly polarized bonds, bonds with bond order larger than 1 and delocalized bonds such as those in aromatic rings should be protected from being cut. In the conventional, nonadaptive QM/MM scheme it is easy to handle this problem, because the QM region is specified at the beginning of the simulation and the user can pick a chemically sensible set of atoms. For our dynamically varying QM (and buffer) regions, chemically sensible decisions have to be made algorithmically. Our implementation allows the user to specify a list of the breakable types of bonds which the software then uses to build the regions automatically.

Finally, as in all force mixing schemes, the abfQM/MM scheme uses dynamical forces that are not conservative, that is they are not the derivatives of a total energy function. This is the price we pay for adaptivity. The nonconservative nature of the dynamics necessitates the use of a thermostat to maintain the correct kinetic temperature throughout the system. The strength of the thermostat we need to use in practice is similar to those that are conventionally used in biomolecular simulations, which suggests that no ill effects will arise purely from the use of a thermostat – the only caveat is that since the use of a thermostat is mandatory, strictly microcanonical simulations cannot be performed. A simple Langevin thermostat is not appropriate in the presence of net heat generation (and would lead to a steady state temperature deviation of several tens of degrees near the QM/MM boundary), so a special *adaptive* thermostat (a combination of Langevin and Nose-Hoover thermostats) that is able to maintain the correct temperature even in the presence of intrinsic heating or cooling is used [310].

10.4.3. Relation to other adaptive QM/MM methods

It is worth noting that the current implementation of abfQM/MM supports the use of several other adaptive QM/MM methods. For example, setting **r_qm_in**, **r_qm_out**, **r_buffer_in** and **r_buffer_out** variables to 0 (for definitions see the next section) leads to the adaptive conventional QM/MM (adconvQM/MM) technique that can be considered also as the zero limit of the adaptive solvent QM/MM (adQM/MM) method [304] without a transition region (see also section 10.3). In this case the *extended* and *reduced* systems are identical and the dynamics is propagated by forces of a convQM/MM calculation whose QM region is adaptive. To save computational time for adconvQM/MM the program first performs the corresponding convQM/MM calculation and then a dummy full MM calculation whose forces are discarded. Another limit can be obtained when **r_buffer_in** and **r_buffer_out** variables are set to 0 (and all other radii are not). This method can be called unbuffered force mixing QM/MM (unbuffQM/MM). It has been observed that the applicability of both adconvQM/MM and unbuffQM/MM depends on several factors (system, QM method, size of *core* / *qm* regions etc.) and it is advised to perform a force convergence test [308, 309] before using them.

10.4.4. Technical glossary

10.4.4.1. Systems

- *extended* system: the first (QM/MM) calculation, which is used for calculating the forces on atoms in the dynamical QM region. To get converged forces on these atoms, a buffer region is added, leading to an extended QM region.
- *reduced* system: the second calculation, which is used for obtaining the MM forces. Either a full MM representation can be used or a QM region that is smaller than the dynamical QM region.

10.4.4.2. Atom types

There are basically four regions in the abfQM/MM method depending on their role during the dynamics: the *core*, the *qm*, the *buffer* and the *mm* regions. These sets are disjoint by definition. There are atoms which are permanent members of a given region and there are others that can change their identity by moving from one region to another. This section describes the different atom types and also gives their name and id used in the implementation. Please note the distinction between the labels “QM” and “*qm*” atoms: the former indicates the QM region used in the actual extended or reduced QM/MM calculations, while the latter is a label used to describe those atoms that, together with the *core* atoms, follow dynamics using quantum mechanical forces.

- *core* atoms (*id* = 1-2): those atoms that constitute the QM region for the reduced system calculation. (The QM atoms in the extended calculation are the *core*, the *qm* and *buffer* atoms together.)
- *user specified core* atoms (*id* = 1, *tag* = CORE_USER): *core* atoms specified by the user. These atoms are permanent *core* atoms during the whole simulation.
    - *core extension* atoms (*id* = 2, *tag* = CORE_EXT): *core* atoms selected by geometrical criteria around the user specified *core* atoms. These atoms belong temporarily to the *core* region.

  $$atom_i \in \{\textit{core extension}\text{ atoms}\} \iff atom_i = f(r_{\text{core\_in}}, r_{\text{core\_out}}, \{\textit{user specified core}\})$$

  $$\{\textit{core}\text{ atoms}\} = \{\textit{user specified core}\text{ atoms}\} \bigcup \{\textit{core extension}\text{ atoms}\}$$

- *qm* atoms (*id* = 3-4): atoms, whose QM forces are used in the MD simulation similarly to *core* atoms but *qm* atoms are excluded from the QM region in the reduced calculation. Their forces are calculated in the extended QM/MM calculation.
    - *user specified qm* atoms (*id* = 3, *tag* = QM_USER): *qm* atoms specified by the user. These atoms are *qm* atoms during the whole simulation or occasionally can become *core extension* atoms.

  - *qm extension* atoms (*id* = 4, *tag* = QM_EXT): *qm* atoms selected by geometrical criteria around the *core* and *user specified qm* atoms. These atoms belong temporarily to the *qm* region.

$$atom_i \in \{\textit{qm extension}\ \text{atoms}\} \iff atom_i = f(r_{\text{qm\_in}}, r_{\text{qm\_out}}, \{\textit{user specified qm}\} \bigcup \{\textit{core}\ \text{atoms}\})$$

$$\{\textit{qm}\ \text{atoms}\} = \{\textit{user specified qm}\ \text{atoms}\} \bigcup \{\textit{qm extension}\ \text{atoms}\}$$

- *buffer* atoms (*id* = 5-6): these atoms are in the buffer region. Although they are treated as QM atoms in the extended calculation, forces on them from this calculation are discarded and they move with forces coming from the reduced calculation in which they are treated with MM.
  - *user specified buffer* atoms (*id* = 5, *tag* = BUFFER_USER): *buffer* atoms specified by the user. These atoms are permanent *buffer* atoms during the whole simulation or occasionally can become *qm* or even *core extension* atoms.
  - *buffer extension* atoms (*id* = 6, *tag* = BUFFER_EXT): *buffer* atoms selected by geometrical criteria around the *qm* and *core* atoms. These atoms belong temporarily to the *buffer* region.

$$atom_i \in \{\textit{buffer extension}\ \text{atoms}\} \iff atom_i = f(r_{\text{buffer\_in}}, r_{\text{buffer\_out}}, \{\text{qm atoms}\} \bigcup \{\text{core atoms}\})$$

$$\{\textit{buffer}\ \text{atoms}\} = \{\textit{user specified buffer}\ \text{atoms}\} \bigcup \{\textit{buffer extension}\ \text{atoms}\}$$

- *mm* atoms (*id* = 7, *tag* = MM): they are MM atoms in both the extended and reduced calculations. For the MD the forces are obtained from the reduced calculation.

- QM atom selections in the reduced and extended QM/MM calculations:

$$\{\text{QM atoms in the reduced system}\} = \{\textit{core}\ \text{atoms}\}$$
$$\{\text{QM atoms in the extended system}\} = \{\textit{core}\ \text{atoms}\} \bigcup \{\textit{qm}\ \text{atoms}\} \bigcup \{\textit{buffer}\ \text{atoms}\}$$

10.4.5. Namelist parameters for adaptive buffer-forced QM/MM simulations

The abfQM/MM implementation requires only two calculations for each MD step, which are performed sequentially (first the computationally more expensive extended then the reduced calculations are carried out). Consequently, unlike adaptive solvent QM/MM (adQM/MM, section 10.3) the subroutines of abfQM/MM are called directly from *sander* and no groupfile is needed. All abfQM/MM related variables should be specified in the *&qmmm* namelist. An example of an abfQM/MM dynamics is shown below:

```
# mdin file - example for adaptive buffered-force QM/MM dynamics
&cntrl
  ...
  ntt=6,    ! adaptive Langevin thermostat is used
  ...
  ifqnt=1,
/

&qmmm
  ...
  abfqmmm=1,              ! activate abf QM/MM
  r_core_in=3.0,          ! inner radius for extended core region
  r_core_out=3.5,         ! outer radius for extended core region
  r_qm_in=3.0,            ! inner radius for extended qm region
  r_qm_out=3.5,           ! outer radius for extended qm region
  r_buffer_in=4.0,        ! inner radius for buffer region
  r_buffer_out=4.5,       ! outer radius for buffer region
  coremask=':1',          ! core region mask
  qmmask=':112, 1129, 1824, 2395',   ! qm region mask
  buffermask='',          ! buffer region mask
  corecharge=0,            ! core region charge
  qmcharge=0,              ! qm region charge
  buffercharge=0,          ! buffer region charge
/
```

10.4.5.1. Basic namelist parameters

abfqmmm	1 activates the adaptive buffered force-mixing method, default is 0 (no abf-QM/MM method is applied).
coremask	*core* atom list specification (in *ambmask* format). Optional, by default (when it is missing or **coremask**=' ') it is an empty set and in this case the reduced system is the full MM representation. Note that at least one of the **coremask** or **qmmask** sets has to be specified.
qmmask	*qm* atom list specification (in *ambmask* format). Optional, by default (when it is missing or **qmmask**=' ') it is an empty set and in this case only atoms in the core region will be treated as QM atoms during the dynamics. Note that at least one of the **coremask** or **qmmask** sets has to be specified.
buffermask	*buffer* atom list specification (in *ambmask* format). Optional, by default (when it is missing or **buffermask**=' ') it is an empty set.
corecharge	Total charge of core atom list defined in **coremask**, default is 0.
qmcharge	Total charge of qm atom list defined in **qmmask**, default is 0.
buffercharge	Total charge of buffer atom list defined in **buffermask**, default is 0.
r_core_in	Inner radius for determining core extension region around user specified core atoms. Default is 0.
r_core_out	Outer radius for determining core extension region around the user specified core atoms. Default is the value specified for **r_core_in**. If **r_core_out < r_core_in** then **r_core_out = r_core_in**.
r_qm_in	Inner radius for determining qm extension region around the core and user specified qm atoms. Default is 0.
r_core_out	Outer radius for determining qm extension region around the core and user specified qm atoms. Default is the value specified for **r_qm_in**. If **r_qm_out < r_qm_in** then **r_qm_out = r_qm_in**.
r_buffer_in	Inner radius for determining buffer extension region around the qm and core atoms. Default is 0.
r_core_out	Outer radius for determining buffer extension region around the qm and core atoms. Default is the value specified for **r_buffer_in**. If **r_buffer_out < r_buffer_in** then **r_buffer_out = r_buffer_in**.

10.4.5.2. Adaptive thermostats' namelist parameters

ntt	Besides the original thermostats in sander, new adaptive ones are also introduced to be able to absorb the heat production due to the nonconservative force-mixing dynamics. The corresponding thermostat can be activated using the **ntt** command. In general, 5 activates the Nose–Hoover (chain)–Langevin, 6 the adaptive Langevin, 7 the adaptive Nose-Hoover chain and 8 the adaptive Nose-Hoover (chain)–Langevin thermostat. For adaptive QM/MM **ntt**=6 or 8 should be used.
gamma_ln	Collision frequency in ps^{-1}
nchain	Number of thermostats in each Nose–Hoover chain of thermostats (default is 1)

10.4.5.3. Miscellaneous namelist parameters

selection_type Type of selection of the different regions. Default is the atom–atom distance based selection (**selection_type** = 1). In this case a given atom is going to belong to an outer region if the distance between the atom in question and any atom in the inner region is less or equal than the corresponding criterion. Option 2 is the flexible sphere selection: for each inner region the radius of the region is calculated (as the largest distance between the centre of mass of the region and any atom belonging to that region), and the distance between the edge of the inner region and the atom in question will determine weather the atom belongs to the outer region or not. Option 3 is fixed sphere based selection: it is the same as option 2 except that only the edge of the innermost region is calculated based on its atoms and then all the other region's borders are calculated geometrically as concentric spheres. For option 2 and 3 the radii of spheres are calculated using the centre region, which is either defined by the user (**centermask**) or it is the **coremask** if specified, otherwise it is **qmmask**. Note that option 2 and 3 selects significantly larger number of atoms than option 1.

initial_selection_type Type of initial selection type. This command controls the initial selection if not an abfqmmm restart is performed (i.e. **read_idrst_file** is not specified). Default is 0, which is the middle sphere selection (i.e. the mean of the corresponding inner and outer radii). Option -1 uses the inner and option 1 applies the outer radius for the first selection.

center_type Type of calculation of center for **selection_type** = 2 and 3. Default is center of mass (option 1), while option 2 is geometric center.

gamma_ln_qm Collision frequency of actual *core* and *qm* atoms in ps^{-1} when adaptive massive Langevin thermostat is applied. Default value is the same as **gamma_ln** defined in &cntrl session.

mom_cons_type Type of force correction for momentum conservation. Default is 1 when the extra force is distributed among the corresponding atoms as equal accelerations. Option 2 applies equal forces on each atom. Options -1/-2 apply an acceleration/force proportional to the absolute value of the current acceleration/force of each atom. The region of atoms where the force correction is distributed is specified by **mom_cons_region**. Option 0 does not apply momentum conservation.

mom_cons_region Specifies the region where the force correction for momentum conservation is distributed. Default is 1 that distributes the correction among only current *core+qm* atoms, option 2 distributes it among current *core+qm+buffer* atoms and option 3 distributes the forces on all atoms. When **mom_cons_region** = 0 the distribution is applied only among *core* atoms.

fix_atom_list > 0 activates the fixed atom list method, default is 0. In fixed atom list mode the different regions are extended only by those solvent molecules that satisfy the given geometrical criteria and no solute atoms will be selected besides the user specified ones in the **coremask**, **qmmask** and **buffermask**. Useful when only solvent exchange is expected.

solvent_atom_number Number of atoms in solvent molecule for fixed atom list mode (**fix_atom_list** > 0), default is 3. Defining this variable is important when the solvent is other than water and the solvent molecule contains more (or less) than 3 atoms.

centermask Centre region atom list specification. Optional, if not defined then it is equal to **coremask**. If **coremask** is neither specified then **centermask** equals to **qmmask**.

oxidation_number_list_file File name of oxidation numbers. Each line in the file must be either a comment (starting by '!' or '#') or a triplet: RES ATOM OXID, where RES can be 'all' (specification for all residues), 'atom' (specification for a given atom), residue name or residue index. If RES $\neq$ 'atom' then ATOM is the atom type name that can be specified completely (e.g. HE2) or partially using '*' (e.g. H* or HE*). If RES = 'atom' then ATOM is the atom index in the topology. OXID is the integer oxidation number. Since different specifications can refer to the same atom, there is a hierarchy of the assignment and the later step always overwrites the previous one: 1. RES = 'all' with partial atom type specification (in the order of X* → XY* → XYZ*), 2. RES = 'all' with complete atom type specification (XYZ1), 3. specified residue type with partial atom type specification, 4. specified residue type with complete atom type specification, 5. residue index with partial atom type specification, 6. residue index with complete atom type specification, 7. atom index specification.

ext_coremask_subset Possible core extension atom set. If specified only those atoms will be chosen according to the corresponding geometrical criteria that can be also found in this list (in the case of fixed atom list method solvent residues having at least one atom in the set will be chosen). If not defined then by default it is the all atom list.

ext_qmmask_subset Possible qm extension atom set. If specified only those atoms will be chosen according to the corresponding geometrical criteria that can be also found in this list (in the case of fixed atom list method solvent residues having at least one atom in the set will be chosen). If not defined then by default it is the all atom list.

ext_buffermask_subset Possible buffer extension atom set. If specified only those atoms will be chosen according to the corresponding geometrical criteria that can be also found in this list(in the case of fixed atom list method solvent residues having at least one atom in the set will be chosen). If not defined then by default it is the all atom list.

cut_bond_list_file File name of breakable bonds for intelligent termination of different regions (core/qm/buffer). Each line in the file must be either a comment (starting by '!' or '#') or a triplet: ATOM1 ARROW ATOM2. ATOM1 and ATOM2 are both either atom types or atom indexes. ARROW specifies the direction of bond breaking: if it is '<=>' then the bond can be split from both directions, if it is '=>' or '<=' then the bond can be cut only from ATOM1 or ATOM2 directions, respectively.

max_bonds_per_atom Maximum number of ligands around any atom in the system. This controls the size of arrays for the intelligent termination. Default is 4 that is good for most biological systems. If there are atoms having more than 4 ligands then adjustment is required.

n_max_recursive Intelligent termination scheme is a recursive subroutine to get a fast and reliable performance. However, it may happen that according to the user specified breakable bonds a very large bond network will be chosen for a given region. To avoid it this variable can be used to control the maximum number of iterations: when the number of iteration reaches the value of **n_max_recursive** the program terminates. Default value is 10000.

min_heavy_mass To keep low the number of atoms in each extension region, by default the geometrical region selection algorithm measures the distances between only heavy atoms, and hydrogen atoms are assigned in a second step according to the heavy atoms they are bonded to. To extend the distance based selection for H atoms as well, decrease the value from its default 4.0 below the atomic mass of hydrogen (e.g. 0.0).

pdb_file File name of a special abfQM/MM related pdb file generated during the dynamics. The first 8 columns have the standard pdb format ('ATOM', atom index, atom name, residue name, residue index, Cartesian coordinates of atom), 9th column is the oxidation number, 10th and 11th columns are the id number and tag according to abfQM/MM implementation, respectively, and the possible following columns include the atom indexes of MM atoms having direct bond to the given atom treated as QM atom in the extended calculation. Default name is *abfqmmm.pdb*.

ntwpdb Frequency of printing out abfQM/MM information into **pdb_file**. Default value is 0 (no printing). Using **ntwpdb** < 0 allows the user to perform a selection test. In this case neither dynamical nor even point calculations are performed, the program terminates after printing the pdb file out.

read_idrst_file Name of abfQM/MM atom id restart file used for restarting simulations. In the beginning of the simulation besides the user specified atoms those become also member of a given region that are within the outer radius. For a given region if the outer radius differs from the inner one, in the beginning of the dynamics the number of atoms will change until it reaches a dynamical equilibrium fluctuation. To avoid this natural transient period in a consecutive restart calculation one can use the **read_idrst_file** generated in the previous run telling the program the abfQM/MM atom id's of the restart configuration. Note that the safe use of **read_idrst_file** requires the same region specifications as in the previous run.

write_idrst_file	Name of abfQM/MM atom id restart file generated during the run. Default name of the file is *abfqmmm.idrst.*
ntwidrst	Frequency of printing the abfQM/MM atom id restart file out. Default is 0 (no printing).
hot_spot	1 activates the hot spot–like adaptive calculation [311] in which the forces of atoms in the buffer region are linear combinations of the forces obtained from the extended and reduced calculations using a smoothing function. Default is 0 (no hot spot–like calculation is performed).

10.5. SEBOMD: SemiEmpirical Born-Oppenheimer Molecular Dynamics

The sander program provides the ability to run SEBOMD (SemiEmpirical Born-Oppenheimer Molecular Dynamics) simulations. During a SEBOMD simulation, all atoms are considered as quantum atoms within the NDDO semiempirical approach (e.g., AM1, PM3, etc). Therefore, unlike QM/MM methods, there is no link atom, no frontier bond, no interaction between any QM and MM atoms (since there is no MM atom). Another consequence of SEBOMD simulations is that the computational time requested to compute energy and forces at each step of a molecular dynamics run can be (very) important. To allow for the computation of “large” systems (i.e., up to a couple of thousands of atoms), an optional linear scaling divide and conquer strategy is implemented[312, 313]. Periodic boundary conditions with long-range electrostatic interactions through Ewald summation can also be applied.

The SEBOMD code implemented in sander is originated from the DivCon program developed in the Merz group while at Pennsylvania State University:

- Steve L. Dixon, Arjan van der Vaart, Valentin Gogonea, James J. Vincent, Edward N. Brothers, Lance M. Westerhoff and Kenneth M. Merz, Jr. *DivCon99*, The Pennsylvania State University, 1999.

Major contributors to the SEBOMD interface are as follows:

- Maintenance, code refactoring, debugging, testing by Gerald Monard
- Original roar interface by Gerald Monard and Arjan van der Vaart[314]
- Original sander port by Jennifer Thomas
- Ewald and Particle Mesh Ewald summation by Laurent Teixidor
- PIF and MAIS semiempirical correction implementation, peptidic corrections by Antoine Marion[315]

10.5.1. Functionalities and limitations

The current SEBOMD implementation allows to run sander simulations with the following functionalities:

- molecular dynamics or energy minimization (`imin = 0, 1, or 5`)
- gas phase or periodic boundary conditions (as defined in the topology file), no support for Generalized Born solvent effect
- For PBC runs, different long range interactions handlers are possible: none, external Particle Mesh Ewald using MM point charges as defined in the topology file, or direct Mulliken Ewald summation.
- temperature regulation as implemented in sander (`ntt` flag)
- pressure regulation: only `barostat = 2` is supported (Monte Carlo barostat)

- parallel implemention (sander.MPI): only the Divide & Conquer approach can be used (`method > 0`)
- available hamiltonians: MNDO, AM1, AM1/d-PhoT, RM1, PM3, PM3/PDDG
- available corrections to PM3 hamiltonians: MAIS and PIF
- as *d-orbitals* are not yet implemented in the SEBOMD code, only the following elements are implemented: H, C, N, O, P, S, F, Cl, Br, I (except for AM1/d-PhoT for which the P element is not yet available because it requires the *d-orbital* implementation)
- maximum number of atoms: 1000; maximum number of residues: 1000
  Note: the SEBOMD code currently uses a static memory allocation as defined in *$AMBERHOME/AmberTools/src/sebomd/sebomd.dim*. Users wishing to simulation bigger systems will have to modify the SEBOMD source code and recompile.

10.5.2. Sample SEBOMD input

To run a SEBOMD calculation, a specific namelist (&sebomd) must be used. It contains all the necessary information for the run. To inform sander that a SEBOMD simulation must be run, two steps are required: 1) switch the ifqnt keyword to 1 (as for a QM/MM calculation); 2) define the qm_theory keyword in the &qmmm namelist to 'SEBOMD'. Here is a sample mdin file for SEBOMD:

```
! example input for SEBOMD simulation
&cntrl
  ...
  ifqnt = 1,             ! switch on QM calculation
/
&qmmm
  qm_theory = 'SEBOMD',  ! use specific SEBOMD routines
/
&sebomd
  hamiltonian = 'AM1',   ! Use the AM1 semiempirical hamiltonian
  charge = 0,            ! total charge on the (full) system is 0
/
```

10.5.3. &sebomd namelist variables

charge = **Integer** Net charge of the system (Default = 0).
Note: SEBOMD only supports closed shell molecular systems.

method Algorithm for the SCF computation.

= 0 (Default) Standard closed-shell algorithm: the Fock matrix is diagonalized at each SCF iteration. (Note: all subsetting parameters are ignored, only one subsystem containing all the atoms will be generated).

= 1 Use linear scaling divide & conquer SCF algorithm. Buffer regions must be specified (dbuff1 and dbuff2). Subsystems are built on an atom-based principle.

= 2 Use linear scaling divide & conquer SCF algorithm. Buffer regions must be specified (dbuff1 and dbuff2). Subsystems are built on an residue-based principle (recommended option over method=1).

= 3 Use linear scaling divide & conquer SCF algorithm. Buffer regions must be specified (dbuff1 and dbuff2). Subsystems are built on an heavy-atom-based principle: each heavy atom plus its hydrogens define one subsystem and there are as many subsystems as the number of non-hydrogen atoms.

ncore **= Integer** When using divide and conquer method (method > 0): specify the number of subsystems used to build the core. (default: ncore = 1)

dbuff1 **= Float** When using divide and conquer method (method > 0): specify the extent of the first buffer region from the core in Å. (default: dbuff1 = 6.0)

dbuff2 **= Float** When using divide and conquer method (method > 0): specify the extent of the second buffer region from the core in Å. (default: dbuff2 = 0.0)

hamiltonian Semiempirical hamiltonian to be used for energy and force calculations. All atoms within the molecular system will be treated at this level of theory. Available semiempirical hamiltonians:

MNDO Request the use of MNDO semiempirical hamiltonian[265]

AM1 Request the use of AM1 semiempirical hamiltonian[263]

PM3 Request the use of PM3 semiempirical hamiltonian (default)[262]

PM3PDDG Request the use of PM3/PDDG semiempirical hamiltonian[266]

RM1 Request the use of RM1 semiempirical hamiltonian[264]

AM1D Request the use of AM1/d-PhoT semiempirical hamiltonian[274]
(Note: phosphorous (P) element is not yet implemented, therefore the AM1D hamiltonian is available only for H, C, N, O, S, F, Cl, Br and I elements)

modif Modification/corrections to the semiempirical energy. Some semiempirical methods have been extended to improve results, mostly in the case of intermolecular interactions. For the moment only PM3 corrections to the energy are available. Possible values are:

none (default) no correction

PIF2 PM3 hamiltonian is modified for **intermolecular** core-core interactions according to the work of Bernal-Uruchurtu et al. and Harb et al. [269, 316–318]. This correction can be applied when using PM3 hamiltonian with a molecular system composed of one (or more) organic molecule(s) in interaction with explicit water molecules. Intermolecular water-water core-core interactions are computed using specific PM3-PIF parameters for aqueous solvent, while intermolecular organic-organic and organic-water intermolecular core-core interactions are computed using another specific set of PM3-PIF parameters. The intermolecular PM3-PIF (PIF2 version) parameters are available only for the following interactions:

	Water		Organic				
	Hw	Ow	H	C	N	O	Cl
Hw	✓	✓	✓	✓	✓	✓	✓
Ow	✓	✓	✓	✓	✓	✓	✓
H	✓	✓	✓	✓	✓	✓	✓
C	✓	✓	✓	∅	✓	✓	∅
N	✓	✓	✓	✓	✓	✓	∅
O	✓	✓	✓	✓	✓	✓	✓
Cl	✓	✓	✓	∅	∅	✓	∅

(✓: intermolecular interaction parameters between the two considered atom types are available; ∅: no intermolecular parameter available)

PIF3 PIF3 is an extension of the PIF2 parameters in which organic hydrogens are distinguished between “hydrophylic” hydrogens and “hydrophobic” hydrogens[315]. In the case of hydrophylic hydrogens, intermolecular interactions between the hydrogen atom and water molecules are computed using PIF2 parameters. In the case of hydrophobic hydrogens, intermolecular interactions between these hydrogen atoms and water molecules are computed using specific parameters. The distinction between hydrophobic and hydrophylic hydrogens is performed using the atom types as specified in the topology file. Hydrogen atom types which are considered as hydrophylic are: H, HO, HS, HW, hn, ho, hp, hs, hw, Ho, hO, hN, and hR. Other hydrogen atom types are considered as hydrophobic.

MAIS1 MAIS extension of the PM3 hamiltonian in which intramolecular **and** intermolecular core-core functions are replaced by specific MAIS functions. This option corresponds to the initial work of Bernal-Uruchurtu *et al.*[268]. Parameters are only available for liquid water (H and O elements).

MAIS2 Second version of the MAIS extension. Parameters are only available for H, O, and Cl elements[269].

longrange Select the type of long range interaction when using periodic boundary conditions:

= 0 (Default) No long range interaction. Only the minimum image convention.

= 1 Perform PME (Particle Mesh Ewald) summation using constant atomic charges extracted from the topology file.

= 2 Perform an Ewald summation using Mulliken atomic charges extracted from the semiempirical wavefunction. Long-range Ewald Mulliken charge effects are incorporated in the Fock matrix of the system to polarize the wavefunction.

dpmax SCF convergence criteria on the density matrix:

= 1e-7 (Default) SCF is considered as converged when density matrix elements between two consecutive SCF steps have not changed more than dpmax. The default value of 1e-7 ensures the conservation of the total energy during NVE simulations. Larger values will speed-up calculations by using less SCF steps but the total energy may not be conserved during molecular dynamics.

fullscf Option to enable pseudo-diagonalization routines

= 0 enable pseudo-diagonalization routine when possible. This can speed-up SCF calculations. (default)

= 1 turn off pseudo-diagonalization. Full diagonalization of the Fock matrix is performed at each iteration of the SCF cycle.

ipolyn Option to activate polynomial interpolation of the guess density matrix

= 0 Use converged density matrix of the previous step as initial (guess) density matrix for the current step. Recommended option for minimization.

= 1 Use polynomial interpolation of the density matrix elements from the last three steps as initial (guess) density matrix for the current step. Recommended option for molecular dynamics runs. (default)

screen verbosity option for SEBOMD calculations

= 0 minimum output. (default)

= 1 output semiempirical energy details at each step

= 2 output semiempirical energy details + the composition of all subsystems when using method > 0.

lambda **= Float** (default 1.0) Enable the computation of a mixed energy value between SEBOMD and full MM computations. If lambda $\neq 1.0$, in addition to a semiempirical calculation, the energy of the full system is evaluated at the MM level. Then energy and forces are mixed according to:

$$Epot = \lambda E(SEBOMD) + (1-\lambda)E(MM)$$

Since, sometimes, semiempirical potential energy surfaces are (very) different from MM surface, the use of the lambda keyword permits to equilibrate MD more easily. For example, from an equilibrated MM system, it is possible to run several SEBOMD simulations using different lambda values from 0.0 (full MM energy) to 1.0 (full QM energy) to obtain an equilibrated SEBOMD simulation.

charge_out	Filename used to save atomic charges. Default = ’sebomd.chg’
ntwc	Every *ntwc* steps, the (Mulliken) atomic charges will be written to the *charge_out* file. If *ntwc* = 0, no atomic charge file will be written. Default = 0. The format of the *charge_out* file is the following: every *ntwc* steps, the energy of the system is first written, then one line per atom is written, containing the x, y, z coordinates and the Mulliken atomic charge of the atom.
peptcorr	flag to apply force field corrections on peptidic bonds Some semiempirical methods do not correctly describe peptidic bond properties, leading to a pyramidal peptide bond nitrogen. An empirical force field correction can be applied to force the planearity of a peptide bond[319]. **= 0** no peptidic correction. (default) **= 1** apply peptidic correction (see Ludwig et al. for details[319])
peptk	**= Float** The force constant of the peptidic correction (in kcal/mol). AM1 default value: peptk = 5.9864 PM3 default value: peptk = 9.8526 MNDO default value: peptk = 6.1737

11. paramfit

Robin Betz

The *paramfit* program allows specific forcefield parameters to be optimized or created by fitting to quantum energy data. *Paramfit* can be used when parameters are missing in the default forcefields and *antechamber* cannot find a replacement, or when existing parameters do not describe the system to the desired level of accuracy, such as for dihedral constants on protein backbones.

Paramfit attempts to make the following statement true: **With the correct AMBER parameters, calculations performed at a quantum level over many conformations of a structures should match those calculated by AMBER.**

Paramfit can calculate the energy of each conformation and/or the force on each atom, and adjust the force field parameters so that these values correspond to input quantum data.

For energies, *Paramfit* attempts to fit the AMBER energy to the quantum energy for a variety of conformations of the input structure, minimizing the equation

$$\sum_{n=1}^{N} w_i \left[\left(E_{MM}(n) - E_{QM}(n) \right)^2 + K \right] = 0$$

where K is a constant that adjusts for different origins in the QM and MM calculations so that minimization may be done to zero and N is the number of molecular conformations that are considered.

For forces, the equation that is optimized is

$$\sum_{n=1}^{N} \sum_{atom=1}^{N_{atoms}} w_i \left| F(n,atom)_{MM} - F(n,atom)_{QM} \right|^2 = 0$$

where the sum of the differences in the forces on each atom should match given the correct set of parameters. Individual structures can be assigned weights $w_i$to give them more or less relative importance in the fit. By default, all weights are set to 1.

The program works by altering the parameters that AMBER uses to describe the molecule, which alter the elements in the AMBER sum that is used to calculate the energy or forces. It is necessary to evaluate over many conformations of the molecule because the parameters should predict how the molecule will behave dynamically rather than statically. To get a good idea of the forces on a dihedral, for example, the energy needs to be evaluated for multiple conformations of the dihedral to see how it changes each time. *Paramfit* will fit so that the energy changes that AMBER predicts will happen when the dihedral twists match the changes predicted with quantum methods.

In order to facilitate force field development, *Paramfit* supports fitting parameters across multiple molecules (for example, fitting a single dihedral backbone term across a variety of input amino acids). Single molecule fits can also be done to generate parameters that are missing or inadequate to describe small molecules or ligands.

Paramfit provides functionality for the majority of steps in the fitting process, including writing input files for quantum packages, specifying which parameters are to be fit, determining the value of K for the system, and finally conducting the fit and saving it in a force field modification file that can be used by other programs. An external quantum program is needed to generate the energies needed for *paramfit* to conduct a fitting. Currently, the program is capable of writing input files for ADF, GAMESS, and Gaussian, although if you write your own input files instead of using *paramfit's* functionality, any quantum package will work.

Paramfit has OpenMP support for parallelization of the AMBER function evaluation over the input conformations, where each core will evaluate the energy for a subset of the conformations. Enable this by adding the *-openmp* option to configure and rebuilding *paramfit.* By default all available cores will be used. To change this, set the `OMP_NUM_THREADS` environment variable to the number of threads to be executed. You will see a speedup directly proportional to the number of cores you are running.

Paramfit now includes several ways fitting functions to aid in parameter generation. It can fit such that the energy of each input structure matches the single-point quantum energies inputted, or can now do the same fitting only with the forces on each atom, which may produce a more accurate fit that is less sensitive to problems with the input structure, and can also fit all dihedral force constants and phases simultaneously to a small set of quantum energies using a method developed by Chad Hopkins and Adrian Roitberg. This method fits every term and requires fewer function evaluations than running the full minimization algorithm, but requires especially good sampling of each torsion angle of interest.

Fitting forces requires several additional options to specify the location of the output forces files in the job control file. The easiest way to create a job control file for any of these options is to use the wizard, which runs automatically when no job control file is specified. This will walk you through the creation of a job control file and write it for you while prompting for all necessary options for the selected fitting function.

It is highly recommended that you fit to single-point quantum energies, as fitting to forces is considerably more expensive in terms of required calculation and still somewhat experimental. The implementation of the dihedral fitting method is requires a varied set of input structures, and does not allow specifying individual dihedrals to be fit. No matter which method is used, please take care to carefully validate all parameters for reasonableness– *paramfit's* fit is dependent on the variation and quality of the input structures and the resulting parameters are not guaranteed in ill-defined areas of the input conformation set. For example, if you fit a dihedral torsion term with input structures sampling the 0-30 degree range of that dihedral, the resulting parameters cannot be expected to give a valid energy of a structure with the dihedral at 90 degrees, as the algorithm merely fits to the available data and cannot make other predictions.

11.1. Usage

Paramfit is called from the command line as follows for a single molecule fit:

```
paramfit -i Job_Control.in -p prmtop -c mdcrd -q QM_data.dat \
-v MEDIUM --random-seed seed
```

Running *paramfit* without any options will run a wizard that assists in the creation of a job control file. It is highly recommended that you use the wizard to assist you in setting run options.

The following switches apply to single molecule fits only:

-p prmtop The molecular topology file for the structure.

-c mdcrd A coordinate file containing many conformations of the input structure. These may be generated by running a short simulation in solution, or by manually specifying coordinates for each atom. It is important that there be a good representation of the solution space for any parameters that are to be optimized– for example, if you want a bond force constant it would be a good idea to have input structures with a good range of values for the length of the that bond type. See Subsection 11.2.6

-q QM_data.dat A file containing the quantum energies of the structures in the coordinate file, in order, one per line. You will have to extract the energies from the output files that the quantum package produces. An example script to do this for Gaussian formatted output files can be found in $AMBERHOME/AmberTools/src/paramfit/scripts.

To fit multiple molecules, the following switches are used:

```
paramfit -i Job_Control.in -pf prmtop_list -cf mdcrd_list -v MEDIUM --random-seed seed
```

Here is a very brief description of the command-line arguments for a multiple molecule fit. For more information on conducting these, fits, please see 11.3.

-pf prmtop_list A file containing a plain-text list of input topology files and the adjustment constant K for each file separated by a space, one per line.

-cf mdcrd_list A file containing a plain text list of input coordinate files, number of structures to read from each file, and directory containing quantum output from each file, separated by a space. These should be specified in the same order as the topologies in the prmtop_list.

The following switches apply to either type of fit:

-i Job_Control.in The job control file for the program. See Section 11.2 for a description of the options and format for this file. If no job control file is specified, a wizard will be initiated that will prompt you for options and help create the file. Use of the wizard is highly recommended when running *Paramfit* for the first time.

-v MEDIUM The verbosity level to run the program at, either LOW, MEDIUM, or HIGH.

–random-seed seed The integer seed for the random number generator. Only specify this parameter when exactly reproducible results are needed for debugging.

11.2. The Job Control File

Similarly to *sander* and other programs, *paramfit* requires a job control file that specifies individual options for each run. The options that apply to your run vary depending on the runtype and the other settings, and they are quite numerous. To aid you in creating a job control file, a wizard has been included that will prompt you about applicable settings and create the job control file for you. Using the wizard is highly recommended, especially when running a fit for the first time. To use the wizard, simply run *paramfit* without any options. **It is highly recommended that you use the wizard to create job control files,** as it prompts for all options relevant to your run and the resulting file can then be easily edited by hand.

The format consists of variable assignments, in the format variable=value, with one assignment per line. Pound signs (#) will comment out lines. See the following sections for a description of what to put in the job control file for various tasks:

11.2.1. General options

paramfit requires several options be set for every run. These variables should usually appear in your job control file.

RUNTYPE Specifies whether this run will be creating quantum input files, setting parameters, or conducting a fit.

- **= CREATE_INPUT** The structures in the coordinate file will be written out as individual input files for a quantum package. See 11.2.2.
- **= SET_PARAMS** Provides an interactive prompt allowing you to specify which parameters will be fit for this molecule. See 11.2.3.
- **= FIT** Conducts a fitting using one of the two minimization algorithms. See 11.2.4for other options that need to be specified.

NSTRUCTURES Specifies how many structures are in the input coordinate file. If this value is less than the total number of structures in the file, only the first **n** will be read. Only applies to single molecule fits! If you are fitting multiple molecules at once, the number of structures for each molecule should be specified in the mdcrd_list file as described in 11.3.

11.2.2. Creating quantum input files

Given a trajectory, *Paramfit* can write input files for a variety of quantum packages. This is necessary to generate the energy values for each input conformation that *Paramfit* will fit to. You do not necessarily need to do this step and can write your own input files if desired. Currently Gaussian, ADF, and GAMESS formats are supported.

Job files will be named sequentially with filename prefix and suffix specified in the job control file. Once all the input files are written, you must run the quantum package yourself. *Paramfit* can read Gaussian output files directly, but for other packages you must extract the energies yourself into a file with one energy per line in the same order as the input structures.

Currently *Paramfit* only supports Gaussian if you are fitting forces, and will read the output files and extract the force information for you. See 11.4 for more information on fitting these.

To enter this mode, set RUNTYPE=CREATE_INPUT and specify the following options in your job control file:

QMHEADER File that will be prepended to all created input files for the quantum program. This specifies things on a per-system basis, such as choice of basis set, amount of memory to use, etc. These parameters will vary depending on which quantum package you are using. Sample header files for all supported quantum packages are included in example_config_files in *paramfit's* source directory.

QMFILEFORMAT Specifies which quantum package the created input files should be formatted for.

= ADF Use the Amsterdam Density Functional Theory package.

= GAMESS Use the General Atomic and Molecular Electronic Structure System (GAMESS).

= GAUSSIAN Use Gaussian.

QM_SYSTEM_CHARGE The integral charge of the system. Defaults to 0. Note that some quantum packages may require this to also be specified in your header file.

QM_SYSTEM_MULTIPLICITY The integral multiplicity of the system. Defaults to 1 (singlet).

QMFILEOUTSTART The prefix for each of the created input files. Defaults to 'Job.' The structure number and then the suffix will be appended to this value.

QMFILEOUTEND The suffix for each of the created input files. Defaults to '.in'. With both default options, the file will be named Job.**n**.in.

11.2.3. Specifying parameters

In order to facilitate batch runs as well as simplify the process of running *paramfit* on larger systems, the parameters to be fit are saved and then loaded in during actual fitting so that they do not have to be specified every time. The parameter setting runtype accomplishes this by prompting whether you would like to fit bond, angle and/or dihedral parameters and then displaying a list of the specific atom types for each so that you can pick exactly what *paramfit* should optimize. This saved file does not specify a value for any of the parameters, but simply indicates which ones are to be changed during fitting.

If you do not wish to save a parameter file, you may instead fit a default set of parameters or be prompted every time. See Subsection 11.2.4.

To enter this mode, set `RUNTYPE=SET_PARAMS` and the following options:

PARAMETER_FILE_NAME Specifies the name of a file in which to store the parameters. When loading these parameters in during a fitting, this line will stay the same. Do not modify this file by hand: *paramfit* numbers each bond, angle, and dihedral in a manner that is consistent but not human-readable.

11.2.4. Fitting options

The fitting function accomplishes the actual parameter modification. It does this by minimizing the least squares difference between the quantum energy and the energy calculated with the AMBER equation over all of the input conformations. For a perfect fit, this means that over all structures, $E_{MD} - E_{QM} + K = 0$.

K is the intrinsic difference between the quantum and the classical energies, which is represented as a parameter that is also fit. The value of K depends on the system, and should be fit once as the only parameter before fitting any other parameters.

To enter this mode, set `RUNTYPE=FIT` and set the following additional variables:

ALGORITHM The minimization algorithm to use. *paramfit* currently implements a genetic algorithm and a simplex algorithm for conduction minimization. Each algorithm requires several parameters and is suited to different problems. Please see 11.2.5 for descriptions of these options and a guide on choosing the appropriate algorithm.

- **= GENETIC**
- **= SIMPLEX**
- **= BOTH** Runs the hybrid genetic algorithm followed by the simplex algorithm to fine tune results
- **= NONE** No fit is performed- useful for calculating energy of each structure with the initial parameters to see their quality

FUNC_TO_FIT The fitting function to use in the calculation.

- **= SUM_SQUARES_AMBER_STANDARD** Standard fit to single-point energies. Recommended selection.
- **= AMBER_FORCES** Fit to the forces on atoms involved in fitted parameters. Currently only supports Gaussian output. See Section 11.4 for details.
- **= DIHEDRAL_LEAST_SQUARES** Use Chad Hopkins and Adrian Roitberg's method to fit all dihedral terms at once. This method will fit all dihedral torsion terms simultaneously with a minimal number of function evaluations, but requires very good sampling of the relevant torsion angles.

K The intrinsic difference between the quantum and classical energies. This value needs to be determined once for each system so that the algorithm can minimize to zero instead of to a constant. See Subsection 11.5.2 for an example.

PARAMETERS_TO_FIT Sets how *paramfit* determines which parameters are to be fit. *paramfit* does not fit electrostatics, but is capable of fitting every other element of the AMBER sum, which include bond harmonic force constant and equilibrium length, angle harmonic force constant and equilibrium angle, and proper and improper dihedral barrier height, phase shift, and periodicity. As a general rule, the fewer parameters there are to fit, the faster and more accurate the results will be. Avoid fitting more parameters than necessary.

- **= DEFAULT** Fit all bond force constants and lengths, angle force constants and sizes, and dihedral force constants. This option will usually fit a very large number of parameters, and is rarely necessary. For most cases, only a few parameters are desired, and they should be fit individually.
- **= K_ONLY** Do not fit any force field parameters. Only fit the value of K (the difference between quantum and classical energies for the system). This needs to be done once per system in order to determine K before any other parameters are fit, as attempting to fit it at the same time results in inaccurate results. Since small changes in K produce a great change in the overall least squares sum, the algorithm will tend to focus on changing the value of K and will neglect the parameters.
- **= LOAD** The list of parameters to be fit is contained in a file that was previously created with the parameter setting runtype. Set PARAMETER_FILE_NAME to the location of this file. To create this file, run *paramfit* with `RUNTYPE=SET_PARAMS`.

SCEE The value by which to scale 1-4 electrostatics for the AMBER sum. Defaults to 1.2

SCNB The value by which to scale 1-4 van der Waals for the AMBER sum. Defaults to 2.0.

QM_ENERGY_UNITS The unit of energy in the quantum data file if you are fitting to energies. This will depend on your quantum package and settings used for the single point calculations.

- **= HARTREE** Default

= KCALMOL

= KJMOL

QM_FORCE_UNITS The unit of force in the quantum data files if you are fitting to forces. This will depend on your quantum package and settings used for the force calculations.

= HARTREE_BOHR Default

= KCALMOL_ANGSTROM

WRITE_ENERGY Saves the final AMBER energy and the quantum data for each structure to the specified file. Plotting these data is useful in verifying the results of the fitting and identifying any problem structures. See Subsection 11.5.3 for more on how to verify the accuracy of results.

WRITE_FRCMOD When the fitting is complete, the parameters will be saved in a force field modification file at this location in addition to displaying them in standard output. This file may be used with leap to create a new *prmtop.* If no value is specified the file will not be created.

SCATTERPLOTS Creates graphs of the bond, angles, and dihedrals found in the input files for each parameter that is being fit. These plots can be visualized using *scripts/scatterplots.sh* found in *paramfit's* source directory. This can be helpful in assessing the quality of the input conformations. No need to specify anything after the = sign for this parameter.

SORT_MDCRDS **= YES** Sorts the input structures in order of increasing energy before conducting the fit. This can aid in identification of problem regions for the initial or fitted parameters, as they may be generally worse on structures in certain energy ranges.

= NO Default

COORDINATE_FORMAT The format of the input coordinate set. *Paramfit* will return an error if the file is in an unexpected format.

= TRAJECTORY Default

= RESTART

11.2.5. Algorithm options

Paramfit implements two minimization algorithms: a simplex and a hybrid simplex-genetic algorithm (GA). The current version of *paramfit* incorporates numerous refinements to the genetic algorithm that require much less input from the user– it is no longer necessary to choose between the simplex or GA. This improved algorithm means that iterative fits are no longer necessary, and the algorithm will converge very close to or at the global minimum on a single run.

The genetic algorithm starts with a randomly generated solution set, which it recombines and alters in ways similar to evolution. The GA will start with many initial randomly generated sets of parameters. It will then determine which are the best by evaluating the AMBER sum, select them for recombination to produce a new set of parameters, randomly alter a few parameters slightly to prevent premature convergence, and iterate. Once several "generations" have passed without improvement, a loosely converging simplex algorithm is run on a random subset of the population, which is then allowed to recover for several generations before further simplex iterations are conducted. This hybrid approach dramatically speeds convergence to the global minimum, while maintaining the strengths of the genetic algorithm in searching a large, complex solution space with low sampling.

The following options in the job control file will control the behavior of the genetic algorithm. In general the default values for these options is sufficient to produce good results, and alterations to them will speed convergence. Options marked *internal algorithm parameter* should not need to be altered by the vast majority of users, as they are already set to their optimum. The algorithm's results should be independent of these values if they are within reasonable ranges (run the wizard for suggestions).

OPTIMIZATIONS The integer number of possible optimizations the algorithm will use. Analogous to the population size in evolution; larger values require more function evaluations and are slower but produce better initial sampling, and smaller ones will delay convergence. Defaults to 50.

SEARCH_SPACE If positive, the algorithm will search for new parameters for everything except dihedral phases within this percentage of the original value, where 1.0 will search within ±100% of the value found in the input *prmtop*. Defaults to searching over the entire range of valid values and ignoring the original value in the topology file. You may wish to alter this value if you know that the original parameters are good and you wish to search in their neighborhood.

MAX_GENERATIONS The maximum number of iterations the algorithm is allowed to run before it returns the best non-converged optimization. Defaults to 50,000. If you find that you repeatedly need to increase this value compared to the default, there are likely significant problems with your system or insufficient input structures.

GENERATIONS_TO_SIMPLEX The number of iterations in a row that must pass without improvement in the best parameter set for simplex refinements to be run on a random 5% of the populations. Set to 0 for a pure genetic algorithm. Smaller values will speed convergence but may result in retrieval of local minima. Defaults to 10.

GENERATIONS_WITHOUT_SIMPLEX The number of generations that must pass between runs of simplex refinement, regardless of improvement in the best parameter set. These iterations serve as a recovery period for the population of the genetic algorithm, and allows time for the simplex results to be incorporated. If set to small or zero values, simplex refinement may run too often, resulting in convergence to a local minima and eliminating the global search properties of the genetic algorithm. Defaults to 10.

GENERATIONS_TO_CONV The number of iterations in a row that must pass without improvement in the best parameter set for the algorithm to be considered converged. Set to a larger value for a longer but potentially more accurate run. Defaults to 50, which is too large for most systems. This counter increments along with the counter to trigger simplex refinement, and at the global minimum simplex refinement will produce no improvement on the population, allowing convergence.

MUTATION_RATE *Internal algorithm parameter* The chance an allele (potential parameter) in the genetic algorithm population has to be randomly set to a new value each generation. Defaults to 0.05.

PARENT_PERCENT *Internal algorithm parameter* The percentage of each generation that is allowed to pass on alleles to the next generation. Defaults to 0.25.

The simplex algorithm is excellent at refining a good set of input parameters, but can converge on physically unreasonable values (such as negative bond force constants) if given a naive guess. For this reason, the genetic algorithm is recommended for finding the global minimum or a close approximation thereof, and the simplex algorithm may be run on the resulting parameters to confirm the results, if desired. The simplex algorithm starts at an initial set of parameters and moves "downhill" iteratively while sampling neighboring areas (much like an amoeba crawling along the function landscape), and converges when the improvement from one step to another becomes negligible. The simplex algorithm is generally faster than the GA, and excels at well-defined systems with a small number of dimensions. This algorithm requires a very well-defined sample space, and the input structures should contain a good range over all the bonds, angles, and dihedrals that are to be optimized. Otherwise, the algorithm tends to wander and will converge in badly defined areas of the sample set. In smaller, well-defined systems with only a few parameters, this algorithm will outperform the genetic algorithm.

Choose the simplex algorithm if you wish to fit only a few parameters and have a large number of input conformations. You may specify the following options to fine-tune the step sizes taken, but for the vast majority of cases the defaults should suffice:

BONDFC_dx Intrinsic length of parameter space for minimization. Used to determine the size of the steps to construct the initial simplex. Should be large enough that the steps sample a sufficiently large area but small enough to not move outside of normal parameter range. Bond force constant step size defaults to 5.0.

BONDEQ_dx Bond equilibrium length step size. Defaults to 0.02.

ANGLEFC_dx Angle force constant step size. Defaults to 1.0.

ANGLEEQ_dx Angle equilibrium step size. Defaults to 0.05.

DIHEDRALBH_dx Dihedral force constant step size. Defaults to 0.2.

DIHEDRALN_dx Dihedral periodicity step size. Defaults to 0.01.

DIHEDRALG_dx Dihedral phase step size. Defaults to 0.05.

K_dx Step size for intrinsic difference constant. Defaults to 10.0.

CONV_LIMIT Floating point number that details the convergence limit for the minimization. The smaller the number, the longer the algorithm will take to converge but the results may be more accurate. Defaults to 1.0E-15, which is very strict.

11.2.6. Bounds Checking

In order to ensure that the algorithms can return meaningful results, bounds checking routines are included in *paramfit*. The bounds checking functionality ensures that the algorithm's results are reasonable given the initial sample set, and also makes sure that the sample set is well-defined.

Since bonds and angles are approximately harmonic, the algorithm's result is reasonable if it lies within a well-defined area of the sample set. Bonds and angle values are therefore checked after the algorithm has finished running. In order to properly fit dihedrals, sample structures should span the entire range of phases for each dihedral that is to be fit. Dihedral checking is therefore accomplished before the algorithm begins to conduct the fit.

Bounds checking defaults to halting execution of the program upon reaching a failing condition. It is not recommended that this behavior be disabled, since the results of the fit are most likely inaccurate. Using the fitted parameters anyway will probably result in an inaccurate depiction of the molecule. Properly represented parameters in the input structures are crucial for a valid fit. Instead of using the parameters, fix the input structures so that data are provided in the missing ranges, which will be stated in the error message, and rerun the program twice: first in CREATE_INPUT mode to obtain quantum energies for the added structures and then in FIT mode to redo the fit.

If you **know** that your input structures describe the parameters to be fit quite well, the selectivity of the bounds checking can be altered by the specifying the following options in the job control file. Use these options with caution, and verify the generated parameters carefully.

CHECK_BOUNDS = ON The recommended and default option. This will halt execution when the bounds check fails.

= WARN Continue upon reaching a bounds failure condition, but output a warning. Do not use the parameters generated by this fit without careful verification! Use the error message and other results to determine if they are reasonable.

BOND_LIMIT Fitting results for bond lengths that are this many Angstroms away from the closest approximation in the input structures will result in a failing condition. Defaults to 0.1.

ANGLE_LIMIT Fitting results for angles that are more than this many radians away from the closes approximation in the input structures will result in a failing condition. Defaults to 0.05π.

DIHEDRAL_SPAN The entire range of valid dihedral angles, 0 to π, for each dihedral that is to be fit should be spanned by this many input structure values, otherwise a failing condition will result. Defaults to 12, meaning that there needs to be a dihedral in every $\frac{\pi}{12}$ radian interval of the valid range.

11.3. Multiple molecule fits

Paramfit supports fitting one or more parameters across multiple molecules, and contains several features to aid in force field development. The program is invoked differently, using a prmtop list and mdcrd list that specify topology and structures for each molecule to fit. Since the value of K is also system-dependent, you will need to fit K for each molecule individually.

Input topologies are specified in a *prmtop list,* which contains the filename of each topology and the value of K for that system, separated by a space. There are no comments permitted in this file. For example:

```
molecule1.prmtop 50.0
molecule2.prmtop 100.0
```

To obtain the value of K for each topology file, conduct a single-molecule fit using all the structures corresponding to that topology and put the resulting value in this file. This enables fitting to zero over multiple molecules.

Input coordinate files are stated in the *coordinate list,* which contains the filename of each coordinate set, the number of structures contained in it, and the filename containing the energy of each structure, separated by a space. Each energy file is exactly the same as single-molecule fits, containing the energy of each structure, one per line, in the same order as the corresponding coordinate file. If there are more structures available in the coordinate file than the number N specified, the first N structures will be used in the fit. An example coordinate list would be:

```
molecule1.mdcrd 200 energy1.dat
molecule2.mdcrd 100 energy2.dat
```

Parameters to fit must be present in all of the available topologies, and the parameter specification file (PARAMETER_FILE_NAME) should be created using a single-molecule invocation of *paramfit.* Saved output files such as energy profile will be named according to the input file name, and a single frcmod will be written if specified. A multiple molecule invocation of *paramfit* uses the following command line options:

```
paramfit -i Job_Control.in -pf prmtop_list -cf mdcrd_list [-v MEDIUM] [--random-seed seed]
```

The only alteration to the job control file necessary for multiple molecule fits is the deletion of the NSTRUCTURES parameter. NSTRUCTURES should not be specified as it is now ambiguous and will result in a program error.

11.4. Fitting Forces

Paramfit can fit to the forces on each atom within an input structure rather than to single point energies. In theory, this provides more data to the fitting algorithm and reduces noise by considering only the forces on atoms involved in a fitted parameter in the function evaluation. This section will walk you through the process of fitting forces using *paramfit.*

Currently, force fitting can only read in Gaussian output files, so input files will be created in the format accepted by that program. Specify in the QMHEADER file the “force” keyword, so Gaussian will print out the forces on each atom, and run *paramfit* in the CREATE_INPUT mode as normal. Then run Gaussian on those input files, keeping the resulting output with the same naming scheme, for example appending “.out” to the name of an input file to indicate its input. For example, in bash:

```
for i in `ls output/Job.*.gjf`; do g09 < $i > $i.out; done
```

To run a fit with forces, you must specify the following options in the job control file, or use the wizard. *Paramfit* will read in the output files from the Gaussian job using the same order and naming scheme, so alter the QM filename parameters so that they match the suffix you appended to Gaussian output files.

```
# Enable force fitting function
FUNC_TO_FIT=AMBER_FORCES
# K irrelevant for force fitting
```

```
K=0.0
# Force units used by Gaussian
QM_FORCE_UNITS=HARTREE_BOHR
# Naming scheme of gaussian output files
QMFILEOUTSTART=output/Job.
QMFILEOUTEND=.gjf.out
```

Specify parameters to fit, algorithm and output options as described previously for fits to energy.

As forces fitting is still experimental, take care to evaluate the resulting parameters.

11.5. Examples

11.5.1. Setting up to fit

The fitting process with *paramfit* follows a specific order. Example job control files for each step and a description of the step follow.

First, write a job control file to create the input structures and run *paramfit:*

```
RUNTYPE=CREATE_INPUT
# Trajectory has 50 structures
NSTRUCTURES=50
# Write in Gaussian format
QMFILEFORMAT=GAUSSIAN
# Prepend this file to QM inputs
QMHEADER=Gaussian.header
$AMBERHOME/bin/paramfit -i Job_Control.in -p prmtop -c mdcrd
```

After all 50 input files have been created, run the quantum program on them. Once it's finished, extract the quantum energies from the output files using the provided script, or write your own. Since the example used Gaussian:

```
$AMBERHOME/AmberTools/src/paramfit/scripts/process_gaussian.x \
output_directory energies.dat
```

Now, or while the quantum jobs are running, since neither the energies nor the structures are needed yet, determine which parameters are to be fit and save them.

```
RUNTYPE=SET_PARAMS
# File to be created
PARAMETER_FILE_NAME=saved_params
$AMBERHOME/bin/paramfit -i Job_Control.in -p prmtop
```

Now the quantum energies to fit have been obtained and the parameters to fit have been set, and the fitting process may begin.

11.5.2. Fitting K

The first step in fitting is determining the value of K for a system. A job control file that will only fit K follows:

```
RUNTYPE=FIT
PARAMETERS_TO_FIT=K_ONLY
# Use the simplex function
FITTING_FUNCTION=SIMPLEX
```

Then,

```
$AMBERHOME/bin/paramfit -i Job_Control.in -p prmtop -c mdcrd -q energies.dat
```

Take this value of K and put it back in the job control file when conducting the actual fit.

```
RUNTYPE=FIT
# Use the parameters specified earlier
PARAMETERS_TO_FIT=LOAD
PARAMETER_FILE_NAME=saved_params
# Genetic algorithm options
FITTING_FUNCTION=GENETIC
OPTIMIZATIONS=500
GENERATIONS_TO_CONV=10
GENERATIONS_TO_SIMPLEX=2
GENERATIONS_WITHOUT_SIMPLEX=5
# Save parameters so they can be read into leap
WRITE_FRCMOD=fitted_params.frcmod
```

And call *paramfit* just as before. This example fit will create a force field modification file that can later be read into *leap* to create a new *prmtop* with the modified parameters for the molecule.

11.5.3. Evaluating Results

When using *paramfit*, it is important to verify the accuracy of the fitted parameters for your input structures. The WRITE_ENERGY option in the Job Control file is useful for this. Set it to a filename and *paramfit* will write the final AMBER energy of each structure next to the quantum energy for the same structure in a file that can be easily graphed.

If you have gnuplot, a script has been provided to quickly show each structure's energies. Assuming your energy file is named energy.dat:

```
$AMBERHOME/AmberTools/src/paramfit/scripts/plot_energy.x energy.dat
```

The resulting graph makes the identification of problem structures much easier, and gives a good visualization of the fit. In general, carefully validate parameters generated by *paramfit* against other data before conducting large simulations.

The SCATTERPLOT option in the job control file can also be useful in assessing the quality of the input structures. If this option is set, *paramfit* will dump a variety of data files indicating the value for all fitted bonds, angles, and dihedrals in the input conformations. These data may be visualized if you have the program gnuplot by running the following command in the directory where *paramfit* was run:

```
$AMBERHOME/AmberTools/src/paramfit/scripts/scatterplots.sh
```

The resulting graphs feature different colored points for each bond, angle, and dihedral type that is being fit for each of the input structures. This is useful in evaluating if the results of the fit are reasonable– for example, if the algorithm converges with an equilibrium bond length that is not similar to any of the structures, that parameter may not be accurate.

Part III.

System preparation

12. Preparing PDB Files

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

The upshot is that most PDB files require some modification before being used in Amber. Most of the recommended steps given below can be achieved with the *pdb4amber* program with the *reduce* option:

```
pdb4amber -i orig.pdb -o new.pdb --reduce --dry
```

This converts the original pdb file into one likely to be more suitable for input into LEaP. But these programs (which are described in Sections 12.4 and 12.5 below) cannot anticipate all situations, so you should still examine the output pdb file to consider the points below.

12.1. Cleaning up Protein PDB Files for AMBER

This is a crucial step in the preparation and many potential problems and subsequent errors arise from omiting this step! (But also note that these are guidelines for beginners: there are certainly circumstances where you may wish to modify the ideas presented here.)

- Analyze the PDB file visually in any viewer that can represent (and maybe modify) the file. Alternatively, use a text editor. Delete all parts which are judged irrelevant for the simulation. Be aware that anything not protein or water will require you to prepare and load extra library files.
- If the x-ray unit cell in the PDB file contains more than one image, choose the entity you want to use and delete the other(s).
- If there is a **ligand**, save it as an MDL standard data file (SDF). Many software packages are able to do this directly. You may also save the ligand in PDB format and then use some other tools later to convert it into a decent SDF file (**including correct bond order and all hydrogens**). It is crucial to **keep the coordinates of its heavy atoms at their original location**. Then delete it from the PDB file. The ligand must treated separately later.
- Delete all water molecules that are not considered relevant. Some waters might be essential for ligand binding. If those waters are kept, they should be made part of the receptor (as distinct "residues"), not of the ligand. *leap* recognizes water if the residue name is WAT or HOH. In later simulations, they may have to be tethered (more or less strongly) to their original positions to prevent them from "evaporating".
- Apply the same delete procedure to ions, co-factors, and other stuff that has no special relevance for the planned simulation.
- **Get rid off all protein (or peptide) hydrogens that are explicitly expressed in the PDB file**. The *reduce* and *leap* utilities adds hydrogens automatically with predefined names. Having hydrogens in PDB files with names that *leap* does not recognize within its residue libraries leads to a total mess.
- Eventually, **remove also all connectivity records**. These are mostly referring to ligands, or, in some cases, to disulfide links. The latter should be explicitly re-connected (see later) without relying on connectivity records in the PDB file.

- The final PDB file of the protein should only contain unique locations for heavy atoms of amino acids (and maybe oxygens of specific water molecules). (In some PDB files, the same amino acid may be represented by different states (conformations). You must decide which unique location you want to use later in the simulations. If you don't do anything, Amber will use the "A" conformation, which is generally the most highly occupied one.) Missing atoms in amino acids are mostly allowed since *leap* can rebuild them if the **residue names** are **correct** and if the **atoms** already present have **correct names** also.
- **Make use of "TER" records to separate parts in the PDB file which are not connected covalently.** This is especially important in protein structures in which parts are missing (gaps). Not separating the loose ends by a "TER" record may lead to strange (and wrong) behavior in *leap* or later in the simulations. Apply the same rule to individual water molecules which you want to keep and separate each water by a "TER" record.

12.2. Residue naming conventions

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

Note also that PDB files written out by *leap* will keep the "special" names, which sometimes leads to annoying effects in software packages which are not prepared for amino acids called HIE, HIP, CYX, and alike. You might consider to change these names back to the standard prior to using these PDB files in other software packages. You can also use the *"-bres"* option in *ambpdb* to do that.

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

Cysteine can exist in free form or as part of a disulfide bridge. PDB residues named "CYS" are automatically converted into a free cysteine with a SH side chain end. If the cysteine is known to be in a **S-S bridge**, the residue name in the PDB file **must** be "**CYX**". In that case, no hydrogen is automatically added to the side chain which ends in a bare sulfur. However, S-S bonds to pairing cysteines are not automatically made but must be specified by the user. The *pytleap* Python script described in section 41.3 takes care of this through a special command line option and a file specifying which residues are to be connected (page 874).

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

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

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

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

Note the atom name "CH3" for this special carbon: another name is not allowed. Hydrogens should be omitted. They are automatically added if the residue name and the heavy atom names are correct.

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

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

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

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

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

The "ACE" residue should be the first residue in a chain (strand) while "NHE" or "NME" should be the last. If cap residues are used to terminate gaps in incomplete protein chains, they must appear at the exact gap location, respecting N-terminal and C-terminal order. Gaps must be separated by a "TER" record in the PDB file. See section 12.3.

12.3. Chains, Residue Numbering, Missing Residues

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

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

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

  The final residue numbers are the ones that must be used in later simulations to refer to individual residues via *AMBER masks* or *NAB atom expressions.* For example, if a protein chain with residues from 30 to 110 is prepared for AMBER simulations, the final numbering will go from 1 to 81. If the original residues 35 to 40 should be fixed or tethered, the actual residues to be specified are 6 to 11. This can lead to serious errors. So be careful about residue numbers. The script *pytleap* described later will always generate a new PDB file with exact AMBER residue numbering and atom names. This PDB file should be used as reference throughout all subsequent AMBER simulations. Above all, when using atom masks or atom expressions (see Appendix 19), always check that they really refer to the desired atoms before running lengthy simulations. *Fixing or tethering wrong atoms are a common error which may easily go unnoticed.*

12.4. pdb4amber

pdb4amber analyses PDB files and cleans them for further usage, especially with the LeaP programs of Amber. The final output files are stripped of everything not directly related to ATOM or HETATM records.

12.4.1. Usage

Typing pdb4amber on the command line without options (or followed by -h) produces the following help message:

```
Usage: pdb4amber [options]

Options:
  --version            show program's version number and exit
  -h, --help           show this help message and exit
  -i FILE, --in=FILE   PDB input file                      (default: stdin)
  -o FILE, --out=FILE  PDB output file                     (default: stdout)
  -y, --nohyd          remove all hydrogen atoms           (default: no)
  -d, --dry            remove all water molecules          (default: no)
  -p, --prot           keep only Amber-compatible residues (default: no)
  --constantph         rename GLU,ASP,HIS for constant pH simulation
  --most-populous      keep most populous alt. conf. (default is to keep first conf.)
  --reduce             run Reduce first to add hydrogens.  (default: no)
  --model=MODEL        model to use from a multi-model pdb file (int). (default: use all models)
```

`--version prints current version and quits`

`-i` or `--in` specifies the PDB input file

`-o` or `--out` specifies the name of the new PDB output file

> NOTE: Input and output file names cannot be the same. Be aware that on some systems with non case-sensitive formats, the automatic check of this requirement might fail if input and output files just differ in upper- and lower-case.

`-y` or `--nohyd` requires that all hydrogens are removed (NOT default). Usually PDB standard PDB files from x-ray have no hydrogens, but already processed PDB files might have hydrogens attached. Since LeaP is strict on hydrogen atom names, it is better to remove all hydrogens prior to running a PDB file through LeaP.

`-d` or `--dry` removes all water molecules (NOT default). Water molecules in PDB files are useful in many occasions. However, you might want to remove them prior to explicit solvent simulations or when running 3D-RISM. Removed waters are stored in a separate file by *pdb4amber* (see under OUTPUT below).

`-p` or `--prot` keeps only the actual protein + water + recognized CAPS like ACE, NME, and NHE (NOT default). It is advised to separate pure protein and ligands prior to preparing files for Amber simulations. With this option, anything not pure protein is removed by pdb4amber and stored in a separate file for further processing later (see under OUTPUT below). Without this option, everything is kept also in the newly generated PDB file.

`--constantph` option to rename GLU, ASP and HIS residues to GL4,AS4 and HIP which is required for constant pH simulations 22

`--reduce` run Reduce 12.5 to add hydrogens to the system

`--most-populous` keep the most populous alternate conformer. The default is to keep the first conformer (usually 'A')

`--model` in a multi-model PDB, keep only the specified model (integer).

pdb4amber also supports command line redirect operators (< for input, > for output) so that it can be used effectively in pipes. Eg. usage:

```
pdb4amber -i pdbin.pdb -o pdbout.pdb
pdb4amber < pdbin.pdb >pdbout.pdb
cat pdbin.pdb | pdb4amber -o pdbout.pdb
```

12.4.2. Output

The new output file (specified with `-o` or `--out`) is a standard PDB file with all residues sequentially renumbered from 1 to N. In addition, several other files are created automatically:

- A text file with the output PDB file name and _renum.txt added. This is a table to help convert the renumbered residues into the original ones.
- A PDB file with the output PDB file name and _nonprot.pdb appended. This is a PDB file that contains only non-protein residues (apart from water), i.e., mainly ligands and other stuff.
- When using `-d` (`--dry`), a PDB file with the output file name plus _water.pdb added. This file contains exclusively the water that has been stripped from the original PDB file.
- A text file with the output PDB file name and _sslink attached, if disulfide bonds have been detected by pdb4amber. This file might be used by the *pytleap* script to generate the correct disulfide bonds between cysteines.

The following information is written to screen, but can also be capturedinto a text file by ending the command line with '2>' e.g.:

```
pdb4amber -i pdbin.pdb -o pdbout.pdb [-options] 2> some_file_name.log
```

Chains: All chain indicators in the PDB file are listed. This is useful especially in cases where the x-ray unit cell contains more than one image of a protein (or complex). In many cases, one is only interested in one main peptide chain. A long list of different chains may indicate that the PDB file should cleaned manually prior to using pdb4amber.

Insertions: Insertions are mostly 'artificial' residue numbers to keep specific key residue numbers in large protein families constant. pdb4amber discards insertion codes and re-numbers all residues from 1 to N. But the insertions are listed to the screen and also included in the _renum.txt file.

Histidines: pdb4amber first checks if the type of each histidine (HIE, HIP, HID) can be determined from explicit hydrogens. Any histidines whose protonation state can not be determined are renamd to HIE. A message alerts the user to all histidine residues (the residue numbers refer to the renumbered scheme!) to allow for manual reassignment if so desired.

Non-standard residues: Non-standard residues (i.e., residues not automatically recognized by Amber) are listed. Mostly they are ligands (sometimes co-factors, detergent, buffer components, etc.). The user must take care of these separately. These residues are also found in the _nonprot.pdb file mentioned above. They are removed from the final output PDB file if the `-p` (`--prot`) option was chosen. Otherwise they are left also in the output PDB file.

Cysteines in disulfide bonds: *pdb4amber* locates possible (most probable) disulfide bonds by checking the distance between SG (gamma sulfur) atoms in cysteines. If a distance SG-SG less than 2.5 Angstrom is found between the SG atoms of two CYS, a disulfide bond is assumed. The respective CYS residues are renamed to CYX (required for Amber) in the final PDB output file. CONECT records are also printed in the final PDB output file which are then automatically recognized by *tleap.* The residue numbers of the CYX residues refer to the renumbered scheme!

Gaps: *pdb4amber* tries hard (and mostly succeeds) in locating 'gaps', i.e., missing residues in the PDB file. This is done by checking distances of consecutive C-N atoms. If such a distance is larger than 2 Angstrom, pdb4amber considers that there is a gap between the two residues and reports the gap to the screen. The listed residue numbers refer to the renumbered scheme! It is up to user to decide how to handle the gaps. *Doing nothing at all will most probably lead to trouble later!* By simply introducing a TER record at the gap, Amber (LeaP) will later introduce the charged N (NH3+) or C (COO-) terminals at the gap borders. If far from the binding site, this might be OK (except in long and unconstrained MD, where such unnatural charges will inevitably lead to unrealistic behavior). The better solution is to introduce ACE or NME caps at the correct positions (in addition to a TER record separating the gap residues). This can be done in various ways (e.g. with PyMol). The correct names of the newly introduced residues (ACE or NME) and atoms (CH3 for the methyl carbon, C, N, O for the others) must be observed!

Missing atoms: pdb4amber tries to determine missing heavy atoms in standard amino acids and reports these. Residue numbers refer to the renumbered sequence. Note that this has no implcations on further usage of the file with LeaP since missing atoms are added automatically anyway. In some cases, this addition may lead to clashes however and it might be useful to know which residues are actually affected by LeaP.

12.4.3. Final remark

The PDB file format and the non-respect of the (in principle well-defined) rules is a constant source of trouble. There is no guarantee that *pdb4amber* works for all possible variants and format violations in PDB files. If you encounter problems with this routine, please report them to me so that I can find a solution. You will never loose (overwrite) your original PDB file. The worst that can happen is that the resulting output is flawed and not usable.

Romain M. Wolf, Basel, December 2013 romain.wolf (at) gmail.com

12.5. reduce

Reduce is a program for adding hydrogens to a Protein DataBank (PDB) molecular structure file. It was developed by J. Michael Word at Duke University in the lab of David and Jane Richardson. *Reduce* is described in: Word, et. al. (1999) Asparagine and Glutamine: Using Hydrogen Atom Contacts in the Choice of Side-chain Amide Orientation, *J. Mol. Biol.* **285**, 1733-1747.

Both proteins and nucleic acids can have hydrogens added. HET groups can also be processed as long as the atom connectivity is provided. A slightly modified version of the connectivity table provided by the PDB is included. The latest version of reduce is available at `http://kinemage.biochem.duke.edu/`.

In most circumstances, the recommended command when using reduce to add hydrogens to a PDB file and standardize the bond lengths of existing hydrogens is

```
reduce -build -nuclear coordfile.pdb > coordfileH.pdb
```

which includes the optimization of adjustable groups (OH, SH, NH3+, Met-CH3, and Asn, Gln and His sidechain orientation). Disulfides, covalent modifications, and connection of the ribose-phosphate nucleic acid backbone, are recognized and any hydrogens eliminated by bonding are skipped. When an amino acid main-chain nitrogen is not connected to the preceding residue or some other group, reduce treats it as the N-terminus and constructs an NH3+ only if the residue number is less than or equal to an adjustable limit (1, by default). Otherwise, it considers the residue to be the observable beginning of an actually-connected fragment and does not protonate the nitrogen. Reduce does not protonate carboxylates (including the C-terminus) because it does not specifically consider pH, instead modeling a neutral environment.

Hydrogens are positioned with respect to the covalently bonded neighbors and these are identified by name. Nonstandard atom names are the primary cause of missing or misplaced hydrogens. If reduce tries to process a file which contains hydrogens with nonstandard names, the existing hydrogens may not be recognized and may interfere with the generation of new hydrogens. The solution may be to remove existing hydrogens before further processing.

There are a number of other, more advance, options for *reduce*, which can be viewed by running:

```
reduce -h
```

13. LEaP

13.1. Introduction

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

LEaP is the basic tool to construct force field files (see Fig. 1.1). 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
Modify internal coordinates within a molecule
Generate files that contain topology and parameters for AMBER and NAB
```

```
usage: tleap [ -I<dir> ] [ -f <file>|- ]
```

The command *tleap* is a simple shell script that calls *teLeap* with a number of standard arguments. Directories to be searched are indicated by one or more "`-I`" flags; standard locations are provided in the *tleap* script. The "`-f`" flag is used to tell tleap to take its input from a file (or from *stdin* if "`-f -`" is specified). If there is no "`-f`" flag, input is taken interactively from the terminal.

A key command for LEaP is loadPdb, which inputs sequence and structure information from Protein Databank Files. *Be sure to read Section 12 for information on how to "clean up" PDB files before loading them.*

13.2. Concepts

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

13.2.1. Commands

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

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

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

For example:

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

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

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

13.2.2. Variables

A variable is a handle for accessing an object. A variable name can be any alphanumeric string whose first character is an alphabetic character. Alphanumeric means that the characters of the name may be letters, numbers, or special symbols such as "*". The following special symbols should not be used in variable names: dollar sign, comma, period (full stop), pound sign (hash), equals sign, space, semicolon, double quote, or the curly braces { and }. LEaP commands should not be used as variable names. Unlike commands, variable names are case-sensitive: "ARG" and "arg" are different variables. Variables are associated with objects using an assignment statement not unlike that found in conventional programming 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 (6.02×10^{23}). 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. This list can be displayed using the `list` command. The contents of a variable can be printed using the `desc` command.

13.2.3. Objects

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

NUMBERs

NUMBERs are simple objects holding double-precision floating point numbers. They serve the same function as "double precision" variables in Fortran and "double" variables in C.

STRINGs

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

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

LISTs

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

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

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

PARMSETs (Parameter Sets)

PARMSETs are objects that contain bond, angle, torsion, and non-bonding parameters for AMBER force field calculations. They are normally loaded from force field data files, such as *parm94.dat*, and frcmod files.

ATOMs

ATOMs are complex objects that do not contain any other objects. The ATOM object corresponds to the chemical concept of an atom. Thus, it is a single entity that may be bonded to other ATOMs and 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 names 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 force field data (**.dat*) or frcmod file. For smooth operation, all atom types must have element and hybridization defined by the `addAtomTypes` command. The standard AMBER force field atom types are added by the selected leaprc file.

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

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

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

RESIDUEs

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

One property of RESIDUEs is connection ATOMs. Connection ATOMs are ATOMs that are used to make linkages between RESIDUEs. For example, in order to create a protein, the N-terminus of one amino acid residue must be linked to the C-terminus of the next residue. This linkage can be made within LEaP by setting the N

ATOM to be a connection ATOM at the N-terminus and the C ATOM to be a connection ATOM at the C-terminus. As another example, two CYX amino acid residues may form a disulfide bridge by crosslinking a connection atom on each residue.

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

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

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

connect2 This defines the third of up to three ATOMs that are used to make links to other RESIDUEs. In amino acids, the convention is that this is the ATOM to which disulfide bridges are made.

restype This property is a STRING that represents the type of the RESIDUE. Currently, it can have one of the following values: "undefined", "solvent", "protein", "nucleic", or "saccharide". Some of the LEaP commands behave in different ways depending on the type of a residue. For example, the solvate commands require that the solvent residues be of type "solvent". It is important that the proper character case be used when defining this property.

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

UNITs

UNITs are the most complex objects within LEaP, and the most important. They may contain RESIDUEs and ATOMs. UNITs, when paired with one or more PARMSETs, contain all of the information required to perform a calculation using AMBER. UNITs can be created using the `createUnit` command. RESIDUEs and ATOMs can be added or deleted from a UNIT using the `add` and `remove` commands. 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. Note that this property is not a UNIT's periodic box for simulating a periodic system. If it is defined as null then no bounding box is defined. If the value is a single NUMBER, the bounding box will be defined to be a cube with each side being *box* Å across. If the value is a LIST, it must contain 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, no solvent cap is defined. If it is a LIST, it must contain four NUMBERs. The first three define the Cartesian coordinates (X, Y, Z) of the origin of the solvent cap in Å, while the fourth defines the radius of the solvent cap, also in Å.

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 Å. The third example defines a solvent cap centered at (15.0, 10.0, 5.0) Å 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.

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 command translates to “describe some RESIDUE named ALA within the UNIT named dipeptide”. The second form translates as “describe 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, either of the following forms of command may be used:

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

Assuming that the ATOM with the name CA has a sequence number 3 within RESIDUE 1, 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.

13.3. Running LEaP

```
xleap -h or tleap -h
```

will give a list of command-line arguments (which are very simple). Once you have started either program, typing “help” will bring up a lot of useful information about possible actions.

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

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 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.

tleap (terminal LEaP) is the non-graphical, command-line-only interface to LEaP. It has the same functionality as the *xleap* main window (Universe Editor Command Window, described below), and uses standard text control keys. *xleap* is a windowing interface to LEaP. In addition to the command-line interface contained in the Universe Editor window, it has a Unit Editor (graphical molecule editor), an Atom Properties Editor, and a Parmset Editor. These editors are discussed in subsequent subsections.

13.3.1. Universe Editor

The window that first appears when the user starts xleap is called the Universe Editor. The Universe Editor is the most basic way in which users can interact with xleap. It has two parts, the "command window," which corresponds to the tleap command interface, and the "pulldown" items above the window, which provide mouse-driven methods to generate specific commands for the command window, either directly or via popped-up dialog boxes. The items in the pulldowns allow the user to generate commands using dialog boxes. To display the "File" pulldown, for example, press the left mouse button on "File;" to select an item in the pulldown, keep the button down, move the mouse to highlight the item, then release the mouse button. A dialog box will then pop up containing fields which the user can fill in, and lists from which values can be chosen; these will be used to generate commands for the command window interface.

13.3.2. 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.

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 nonzero 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 mouse pointer within the viewing window and typing control-C. Since only internal coordinates are energy minimized, steric overlap can result.
- "Edit selected atoms" – pops up an Atom Properties Editor, a tool for examining/setting the properties of the selected ATOMs. The Atom Properties Editor allows the user to edit the ATOM names, types and charges in a convenient table format. It is described in a separate subsection below.
- "Flip chirality" – This command inverts the chirality of all selected ATOMs. In order for the chirality to be inverted, the ATOM cannot be in more than one ring. The operation causes the lightest chains leaving the ATOM to be moved so as to invert the chirality. If the ATOM has only three chains attached to it, then only one of the chains will be moved.
- "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.
- "Types" – toggles display of molecular mechanics atom types. The ATOM types are displayed within parentheses "()".
- "Charges" – toggles display of the atomic charges.
- "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.

Unit Editor manipulation buttons

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

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

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

Twist *Twist* mode operates on previously-*Select*ed atoms. The intention is to allow rotation about dihedrals; if too many atoms are selected, odd transformations can occur. While in the *Twist* mode, the mouse pointer looks like a curved arrow. Twisting is driven by holding down the left-button anywhere in the viewing window and moving the mouse up and down. It is important to select a complete torsion (all four atoms) before trying to "twist" it.

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

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

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

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

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

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

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.

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

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

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

13.3.3. 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, and element.

13.3.4. 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.

13.4. Basic instructions for using LEaP to build molecules

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

13.4.1. Building a Molecule For Molecular Mechanics

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

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

The most typical command sequence is the following:

```
source leaprc.protein.ff14SB (load a force field)
x = loadPdb trypsin.pdb (load in a structure)
.... add in cross-links, solvate, etc.
saveAmberParm x prmtop prmcrd (save files)
```

There are a number of variants of this:

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

2. Be very attentive to any errors produced in the `loadPdb` step; these generally mean that LEaP has misread the file. A general rule of thumb is to keep editing your input PDB file until LEaP stops complaining. It is often convenient to use the `addPdbAtomMap` or `addPdbResMap` commands to make systematic changes from the names in your PDB files to those in the Amber topology files; see the leaprc files in *$AMBERHOME/dat/leap/cmd* for examples of this. *Be sure to read Section 12 for information on how to "clean up" PDB files before loading them.*

3. The `saveAmberParm` command cited above is appropriate for most force fields; for polarizable calculations you will need to use `saveAmberParmPol`.

13.4.2. Amino Acid Residues

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.

Histidine can exist either as the protonated species or as a neutral species with a hydrogen at the δ or ε 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 HIE. Thus, if a PDB file is read that contains the residue HIS, the residue will be assigned to the HIE UNIT object. This feature can be changed within one's own leaprc file.

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

13.4.3. Nucleic Acid Residues

The "D" prefix can be used to distinguish between deoxyribose and ribose units. Residue names like "A" or "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 (nucleosides) capped by hydrogens end their names with "N", as in "DAN".

13.5. 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 square brackets (e.g., `[arg]`), then that argument is optional. When examples are shown, the command line is prefaced by "> ", and the program output is shown without this character preface.

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

13.5.1. 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.

```
> 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.
```

13.5.2. addAtomTypes

```
addAtomTypes { { type element 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.

13.5.3. addIons and addIons2

```
addIons unit ion1 numIon1 [ion2 numIon2]
addIons2 unit ion1 numIon1 [ion2 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* must not be specified. If solvent is present, it is ignored in the charge and steric calculations, and if an ion has a steric conflict with a solvent molecule, the ion is moved to the center of that solvent molecule, and the latter is deleted. (To avoid this behavior, either solvate _after_ `addions`, or use `addIons2`.) Ions must be monatomic. This procedure is not guaranteed to globally minimize the electrostatic energy. When neutralizing regular-backbone nucleic acids, the first cations will generally be placed between phosphates, leaving the final two ions to be placed somewhere around the middle of the molecule. The default grid resolution is 1 Å, extending from an inner radius of (*maxIonVdwRadius* + *maxSoluteAtomVdwRadius*) to an outer radius 4 Å beyond. A distance-dependent dielectric is used for speed. `addIons2` is the same as `addIons`, except solvent and solute are treated the same.

13.5.4. addIonsRand

```
addIonsRand unit ion1 #ion1 [ion2 #ion2] [separation]
```

Adds counterions in a shell around *unit* by replacing random solvent molecules. If *#ion1* is 0, the unit is neutralized (*ion1* must be opposite in charge to *unit*, and *ion2* cannot be specified). Otherwise, the specified numbers of *ion1* [*ion2*] are added [in alternating order]. If *separation* is specified, ions will be guaranteed to be more than that distance apart in Angstroms.

Ions must be monoatomic. This procedure is much faster than `addIons`, as it does not calculate charges. Solvent must be present. It must be possible to position the requested number of ions with the given separation in the solvent.

13.5.5. addPath

```
addPath path
```

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

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

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

13.5.6. addPdbAtomMap

```
addPdbAtomMap 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 startup file, “leaprc”, so that assignments are made at the beginning of the session. *list* should be a LIST of LISTs. Each sublist should contain 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.

13.5.7. addPdbResMap

```
addPdbResMap 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 startup 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 string1 string2 }
```

where *double* can be 0 or 1, *string1* is the name within the PDB file, and *string2* is the variable name to which *string1* 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:

```
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 } : :
> }
```

13.5.8. alias

```
alias [ string1 [ 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 that string will be removed from the Alias Table. If no arguments are given to 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 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.

13.5.9. bond

```
bond atom1 atom2 [ order ]
```

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

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

13.5.10. bondByDistance

```
bondByDistance container [ maxBond ]
```

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

```
bondByDistance alkylChain
```

13.5.11. check

```
check unit [ parms ]
```

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

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

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

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

13.5.12. combine

```
variable = combine 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>
```

13.5.13. copy

```
newvariable = copy variable
```

In most cases, creates an exact duplicate of the object *variable*. Since *newvariable* is not pointing to the same object as *variable*, changing the contents of one object will not alter the other object. Example:

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

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

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

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

Note that in a few instances, the copy command does not produce an exact copy. This is particularly relevant when making copies of oligosaccharide residues. In these, the copy command invariably inverts chirality at the anomeric carbon. The workaround for this is to use the copy command twice, where the second call inverts the chirality back.

13.5.14. createAtom

```
variable = createAtom name type 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.)

13.5.15. createResidue

```
variable = createResidue name
```

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

13.5.16. createUnit

```
variable = createUnit name
```

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

13.5.17. deleteBond

```
deleteBond atom1 atom2
```

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

13.5.18. desc

```
desc variable
```

Print a description of the object *variable*. 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
```

13.5.19. groupSelectedAtoms

```
groupSelectedAtoms unit name
```

Create a group within *unit* with the name *name*, using all of the ATOMs within *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 LISTs 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.)

13.5.20. help

```
help [string]
```

This command prints a description of the command in *string*. If no argument is given, a list of help topics is provided.

13.5.21. impose

```
impose unit seqlist internals
```

The `impose` command allows the user to impose internal coordinates on *unit*. The list of RESIDUEs to impose the internal coordinates upon is in *seqlist*. The internal coordinates to impose are in *internals*, which is an object of type LIST.

The command works by looking into each RESIDUE within *unit* that is listed in *seqlist* 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. A range of sequence numbers is 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 *unit*, in which case the range will stop at the beginning or end of *unit* as appropriate. For example, the range { 1 999 } will include 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 α-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 Å between the α and β carbon atoms. RESIDUEs without an ATOM named CB, such as glycine, will be unaffected.

It is important to understand that the impose command attempts to perform the intended action on all residues in the *seqlist*, but does not necessarily limit itself to acting only upon *internals* contained within those residues. That is, the list does not limit the residues to consider. Rather, it is a list of all starting points to consider. In other words, to specify a *seqlist* of { 3 4 } tells *impose* to attempt to set two torsions, one starting in residue 3 and the other starting in residue 4. It does not specify that the torsion should only be set if the atoms are found within residues 3 and/or 4.

Because of this, one must be careful when setting torsions between two residues. It is necessary to know which atoms are contained in which residues. Consider the following trisaccharide:

$$\alpha\text{-D-Glc}p\text{-(1-6)-}\beta\text{-D-Man}p\text{-(1-6)-}\beta\text{-D-Gal}p\text{-OH}$$

To build it most simply in leap requires the following directive. Note that the build order in leap is the reverse of the standard order in which the residues are written above.

```
glycan = sequence { ROH 6LB 6MB 0GA }
```

A proper build of a 1-6 oligosaccharide linkage often requires setting three torsions. In the manner that residues are defined in the Glycam force fields, the atoms describing two of those torsions, ϕ and ψ, span two residues. However, the atoms in the third, ω, exist entirely within one residue. In fact, they exist within all three glycan residues in the example above. The following commands will set only the three torsions in the glycosidic linkage between residues 4 (0GA) and 3 (6MB).

```
impose glycan { 4 } { { "H1" "C1" "O6" "C6" -60.0 } } # O6 & C6 are in residue 3
impose glycan { 4 } { { "C1" "O6" "C6" "C5" 180.0 } } # only C1 is in residue 4
impose glycan { 3 } { { "O6" "C6" "C5" "O5" 60.0 } } # all are in residue 3
```

The common misconception that the *seqlist* sets a limit on the residues affected can cause trouble in this case. For example, this command

```
impose glycan { 4 3 } { { "H1" "C1" "O6" "C6" -60.0 } }
```

will find all sequences beginning in residue 4 and in residue 3 that contain the serially bonded atoms H1 C1 O6 and C6. Therefore, in this case, it will set the specified torsions between residues 4 and 3 as well as between 3 and 2. Similarly, this command

```
impose peptide { 4 } { { "O6" "C6" "C5" "O5" 60.0 } }
```

will not affect any inter-residue linkage, but instead will set the C5-C6 torsion in the glucopyranoside (0GA) at the non-reducing end of the oligosaccharide.

The ordering and content within the *internals* list is important as well. For these examples, consider the simple peptide sequence:

```
peptide = sequence { ALA ALA ALA ALA }
```

The ordering of the *internals* specifies the atoms to which the torsion set is applied. The *impose* command will find the first atom in the *internals* list, check for the presence of a bonded second atom, and so forth. It will then apply the action, here a torsion, to those four atoms. For example, this command:

```
impose peptide { 3 } { { "N" "CA" "C" "N" -40.0 } }  # between 3 and 4
```

will set the torsion between residues 3 and 4. However, this one:

```
impose peptide { 3 } { { "N" "C" "CA" "N" -40.0 } }  # between 3 and 2
```

will set the torsion between residues 3 and 2.

If at any point, the *impose* command does not find an atom bonded to a previous atom in an *internals* list, it will silently ignore the command. This is likely to occur in two instances. One, the atom simply might not exist in the residue:

```
impose peptide { 3 } { { "N" "CA" "CB" "HB4" 10.0 } }  # no effect, silent
```

Here, of course, there is no atom named HB4 in alanine. Similarly, improper torsions are ignored. For example, this command also has no effect:

```
impose peptide { 3 } { { "N" "HB1" "CA" "CB" 10.0 } }  # no effect, silent
```

because HB1 is not bonded to N.

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 is applied to each RESIDUE.

It is also important to note that the impose command performs its actions entirely using internal coordinates. Because of this, it is difficult to predict the resulting behavior when the coordinates are translated back to cartesian, for example when writing a PDB file.

13.5.22. 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:

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

13.5.23. loadAmberParams

```
variable = loadAmberParams filename
```

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

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

13.5.24. loadAmberPrep

```
loadAmberPrep filename [ 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* (a STRING) is provided, its contents 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.

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

13.5.25. loadOff

```
loadOff 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
```

13.5.26. loadMol2

```
variable = loadMol2 filename
```

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

13.5.27. loadPdb

```
variable = loadPdb filename
```

Load a Protein Data Bank (PDB) format file with the file name *filename* into *variable*, a UNIT. 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
```

13.5.28. loadPdbUsingSeq

```
loadPdbUsingSeq filename unitlist
```

This command reads a PDB format 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”.

13.5.29. logFile

```
logFile 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
```

13.5.30. measureGeom

```
measureGeom atom1 atom2 [ atom3 [ 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 ....
```

13.5.31. quit

Quit the LEaP program.

13.5.32. remove

```
remove a b
```

Remove the object *b* from the object *a*. If *a* does not contain *b*, 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 any other variable name, 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>
```

13.5.33. saveAmberParm

```
saveAmberParm unit topologyfilename coordinatefilename
```

Save the Amber/NAB topology and coordinate files for *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 *unit*. It produces topology files and coordinate files that are identical in format to those produced by Amber PARM and can be read into Amber and NAB 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
```

13.5.34. saveMol2

```
saveMol2 unit filename type-flag
```

Write *unit* to the file *filename* as a Tripos mol2 format file. If *type-flag* is 0, the Tripos (Sybyl) atom types will be used; if *type-flag* is 1, the Amber atom types present in *unit* will be used. Generally, you would want to set *type-flag* to 1, unless you need the Sybyl atom types for use in some program outside Amber; Amber itself has no force fields that use Sybyl atom types.

13.5.35. saveOff

```
saveOff object 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 within the *object* argument. If the file *filename* already exists, the new objects will be added to it. 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.)

13.5.36. savePdb

```
savePdb unit 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
```

13.5.37. sequence

```
variable = sequence list
```

The `sequence` command is used to combine the contents of *list*, which should be a LIST of UNITs, into a new, single UNIT. This new UNIT is constructed by taking each UNIT in *list* 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 }
```

13.5.38. set

This command operates in two modes. In the first, it sets default values for some parameters. In the second, it sets specific properties to containers (for example, UNITs).

Defaults can be set in LEaP for the global parameters below with this usage:

```
set default parameter value
```

For example:

```
set default PBRadii mbondi
```

OldPrmtopFormat If set to "on", the `saveAmberParm` command will write a prmtop file in the format used in Amber 6 and earlier versions; if set to "off" (the default), it will use the new format. This is discouraged for general use and is available mainly for backwards compatibility with programs that expect old-style topology files or for testing.

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

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

PBRadii Used to choose various sets of atomic radii for generalized Born or Poisson-Boltzmann calculations. Options are: "bondi", which gives values from Ref. [284], which should be used with *igb* = 7; "mbondi", which is the default, and the recommended parameter set for *igb* = 1 [154]; "mbondi2", which is a second modification of the Bondi radii set [138], and should be used with *igb* = 2 or 5; "mbondi3", which is a third modification of the Bondi radii set [20] recommended for use with igb = 8; and "amber6", which is only to be used for reproducing very early calculations that used *igb* = 1 [136].

nocenter If set to “on”, LEaP will not center the coordinates inside the box for a periodic simulation, but will leave them unchanged (as it does for non-periodic simulations); if set to “off” (the default), centering of coordinates will take place (as it always has, in previous versions of LEaP). Avoiding coordinate translations can be useful to avoid changing reference (perhaps experimental) coordinates. This option may be especially helpful for crystal simulations.

reorder_residues If set to “off” residues in the output will be left in the same order they were found in the input file. The default behavior (“on”) is to place non-solvent residues first, followed by solvent residues, followed by solvent cap residues (if cap exists). “off” can, for example, be useful in crystal simulations (keep residues belonging to each asymmetric unit separate), but note that turning residue ordering off is untested and may lead to unforseen behavior. Only set to “off” if you know what you are doing!

The parameters listed below can be set for the specified *containers* within LEaP using the following syntax:

```
set container parameter object
```

Some examples:

```
set ATOM name "name"
set RESIDUE connect0 ATOM
my_system = loadPDB file.pdb
set my_system box {25 30 32}
```

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 This STRING is a unique identifier for an ATOM in its final state during a Free Energy Perturbation calculation. This functionality is no longer implemented in Amber.

pertType This STRING is the AMBER force field atom type of a perturbed ATOM. This functionality is no longer implemented in Amber.

pertCharge This NUMBER represents the final electrostatic point charge on an ATOM during a Free Energy Perturbation. This function is no longer implemented in Amber.

For RESIDUEs:

connect0 This identifies the first of up to three ATOMs that will be used to make links to other RESIDUEs. In a UNIT containing a single RESIDUE, the RESIDUE’s connect0 ATOM is usually defined as the UNIT’s head ATOM.

connect1 This identifies the second of up to three ATOMs that will be used to make links to other RESIDUEs. In a UNIT containing a single RESIDUE, the RESIDUE’s connect1 ATOM is usually defined as the UNIT’s tail ATOM.

connect2 This identifies the third of up to three ATOMs that will be used to make links to other RESIDUEs. In amino acids, the convention is that this is the ATOM to which disulfide bridges are made.

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

name This STRING property is the RESIDUE name.

For UNITs:

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

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

box This property defines the bounding box of the UNIT (*container*). If *object* is set to null then no bounding box is defined. If it is a single NUMBER, the bounding box will be defined to be a cube with each side being NUMBER Å across. If it is a LIST, it must contain three NUMBERs, the lengths (in Å) of the three sides of the bounding box. Note that this box is not a UNIT's periodic box for simulating a periodic system. See the `setBox` and `solvate` family of commands to add a periodic box to a UNIT.

cap This property defines the solvent cap of the UNIT. If it is set to null then no solvent cap is defined. Otherwise, it should be a LIST of four NUMBERs; the first three NUMBERs define the Cartesian coordinates (X, Y, Z) of the origin of the solvent cap in Å, while the fourth defines the radius of the solvent cap, also in Å.

13.5.39. setBox

```
setBox solute enclosure [ distance ]
```

This command creates a periodic box around *solute*, which should be a UNIT. It does not add any solvent to the system. `setBox` creates a cuboid box. The *enclosure* parameter determines whether the box encloses entire atoms or just atom centers. The former case is specified by the STRING value "vdw" for *enclosure* and the latter case by the STRING "centers". Use "centers" if the system has been previously equilibrated as a periodic box. The minimum distance between any atom in *solute* and the edge of the periodic box is given by the *distance* parameter; see the solvateBox command for more details.

```
> mol = loadpdb my.pdb
> setBox mol "vdw"
```

13.5.40. solvateBox and solvateOct

```
solvateBox solute solvent distance [ "iso" ] [ closeness ]
solvateOct solute solvent distance [ "iso" ] [ closeness ]
```

These two commands create periodic solvent boxes around *solute*, which should be a UNIT. `solvateBox` creates a cuboid box, while `solvateOct` creates a truncated octahedron. *solute* is modified by the addition of copies of the RESIDUEs found within *solvent*, which should also be a UNIT, such that the minimum distance between any atom originally present in *solute* and the edge of the periodic box is given by the *distance* parameter. The resulting solvent box will be repeated in all three spatial directions.

If the distance parameter is a single NUMBER then the minimum distance is the same for the x, y, and z directions, unless the STRING "iso" parameter is specified to make the box or truncated octahedron isometric. For solvateBox if "iso" is used, the solute is rotated to orient the principal axes, otherwise it is just centered on the origin. For solvateOct if the "iso" option is used, the isometric truncated octahedron is rotated to an orientation used by the PME code, and the box and angle dimensions output by the saveAmberParm* commands are adjusted for PME code imaging.In solvateBox, if the distance parameter 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.In solvateOct, when a LIST is given for the distance parameter, four numbers are given instead of three, where the fourth is the diagonal clearance. If 0.0 is given as the fourth number, the diagonal clearance resulting from the application of the x,y,z clearances is reported. If a non-0 value is given, this may require scaling up the other clearances, which is also reported. Similarly, if a single NUMBER is given, any scaleup of the x,y,z buffer to accommodate the diagonal clip is reported.

The optional *closeness* parameter can be used to control how close, in Å, solvent ATOMs may come to solute ATOMs. The default value of *closeness* 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 and its nearest solute ATOM is less than the sum of the two ATOMs' van der Waals radii multiplied by *closeness*.

```
> mol = loadpdb my.pdb
> solvateOct mol TIP3PBOX 12.0 0.75
```

13.5.41. solvateCap

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

The `solvateCap` command creates a solvent cap around *solute*, which is a UNIT. *solute* is modified by the addition of copies of the RESIDUEs found within *solvent*, which should also be a UNIT. The solvent box will be repeated in all three spatial directions to create a large solvent sphere with a radius of *radius* Å.

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

The optional *closeness* parameter can be used to control how close, in Å, solvent ATOMs may come to solute ATOMs. The default value of *closeness* 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 and its nearest solute ATOM is less than the sum of the two ATOMs' van der Waals radii multiplied by *closeness*.

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

```
> mol = loadpdb my.pdb
> solvateCap mol TIP3PBOX mol.2.CA 12.0 0.75
```

13.5.42. solvateShell

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

The `solvateShell` command adds a solvent shell to *solute*, which should be a UNIT. *solute* is modified by the addition of copies of the RESIDUEs found within *solvent*, which should also be a UNIT. The resulting solute/solvent UNIT will be irregular in shape since it will reflect the contours of the original solute molecule. 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* Å thick. Solvent RESIDUEs are then added to *solute* if they lie within the shell defined by *thickness* and do not overlap with any ATOM originally present in *solute*. The optional *closeness* parameter can be used to control how close solvent ATOMs can come to solute ATOMs. The default value of the *closeness* argument is 1.0. Please see the `solvateBox` command for more details on the closeness parameter.

```
> mol = loadpdb my.pdb
> solvateShell mol TIP3PBOX 12.0 0.8
```

13.5.43. source

```
source filename
```

This command executes the contents of the file given by *filename*, treating them as LEaP commands. To display the commands as they are read, see the `verbosity` command.

13.5.44. transform

```
transform atoms, matrix
```

Transform all of the ATOMs within *atoms* by a symmetry operation. The symmetry operation is represented as a (3 × 3) or (4 × 4) matrix, and given as nine or sixteen NUMBERs in *matrix*, a LIST of LISTs. 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 6 Å 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 } }
```

13.5.45. translate

```
translate atoms direction
```

Translate all of the ATOMs within *atoms* by the vector given by *direction*, a LIST of three NUMBERs.

Example:

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

13.5.46. verbosity

```
verbosity 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
```

13.5.47. zMatrix

```
zMatrix object zmatrix
```

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*, an ATOM, by placing it *bond12* Å along the x-axis from ATOM *a2*. *a2* is placed at the origin if its coordinates are not defined.

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

This entry defines the coordinate of *a1* by placing it *bond12* Å away from *a2* making an angle of *angle123* degrees between *a1*, *a2* and *a3*. The angle is measured in a right-hand sense and in the xy 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* Å away from *a2*, creating an angle of *angle123* degrees between *a1*, *a2*, and *a3*, and making a torsion angle of *torsion1234* degrees 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* Å away from *a2*, and making angles *angle123* degrees between *a1*, *a2*, and *a3*, and *angle124* degrees between *a1*, *a2*, and *a4*. The argument orientation defines whether *a1* is above or below a plane defined by *a2*, *a3* and *a4*. If *orientation* is positive, *a1* will be placed so that the triple product $((a3-a2) \times (a4-a2)) \cdot (a1-a2)$ 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* and *a4*) are either ATOMs, or STRINGs containing names of ATOMs that already exist within *object*. The subsequent arguments (*bond12*, *angle123*, *torsion1234* or *angle124*, and *orientation*) are all NUMBERs. Any ATOM can be placed at the *a1* position, even one that has 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.)

13.6. Building oligosaccharides, lipids and glycoproteins

Build assistance available at GLYCAM-Web:

The approaches presented below have been automated, with many additional options available, at the GLYCAM-Web site: www.glycam.org. The capabilities of the website are being expanded. Currently, the available functionalities include:

```
Oligosaccharides, linear and branched
Glycoproteins, O- or N-linked, with multiple glycans
Builds of oligosaccharides via URL directive
```

Build assistance available in the AmberTools tests:

Examples in addition to those described below can be found in the AmberTools tests. The relevant files are located in:

```
$AMBERHOME/AmberTools/test/leap/glycam  # main test directory
$AMBERHOME/AmberTools/test/leap/glycam/06EPb  # extra points oligosaccharides
$AMBERHOME/AmberTools/test/leap/glycam/06j  # main oligosaccharides
$AMBERHOME/AmberTools/test/leap/glycam/06j_10  # glycoprotein with ff10
$AMBERHOME/AmberTools/test/leap/glycam/06j_12SB  # glycoprotein with ff12SB
```

PLEASE NOTE: The molecules in the test directories were constructed for the purpose of testing functionality in AmberTools. They might not be ready for simulations as they are. Some might be in configurations with severe clashes. Most structural issues can be resolved by manipulating appropriate torsions. The glycoprotein tests contain usage examples for torsion manipulations using the impose command.

Each sub-directory below "glycam" contains tests relevant to specific force fields. To run an individual test, saving all relevant output and intermediate files, change to the sub-directory and issue the command:

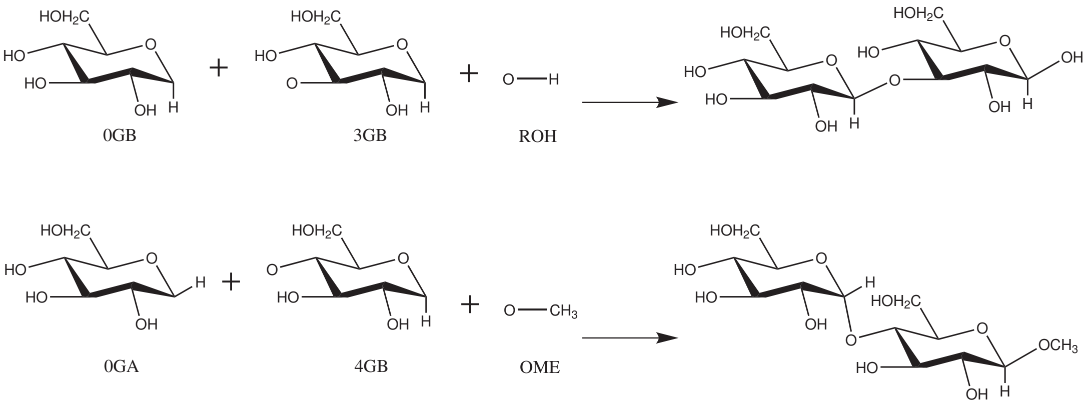


Figure 13.1.: *Schematic representation of disaccharide formation, indicating the need for open valences on carbon and oxygen atoms at linkage positions.*

```
./Run.glycam evaluate
```

To return the directory to its previous state, run:

```
./Run.glycam clean
```

The 00_README file in the main directory contains more information about using the tests.

Additional notes about this section:

Before continuing in this section, you should review the GLYCAM naming conventions covered in Section 3.3. After that, there are two important things to keep in mind. The first is that GLYCAM is designed to build oligosaccharides, not just monosaccharides. In order to link the monosaccharides together, each residue in GLYCAM will have at least one open valence position. That is, each GLYCAM residue lacks either a hydroxyl group or a hydroxyl proton, and may be lacking more than one proton depending on the number of branching locations. Thus, none of the residues is a complete molecule unto itself. For example, if you wish to build α-D-glucopyranose, you must explicitly specify the anomeric -OH group (see Figure 13.1 for two examples).

The second thing to keep in mind is that when the `sequence` command is used in LEaP to link monosaccharides together to form a linear oligosaccharide (analogous to peptide generation), the residue ordering is opposite to the standard convention for writing the sequence. For example, to build the disaccharides illustrated in Figure 13.1, using the sequence command in LEaP, the format would be:

```
upperdisacc = sequence { ROH 3GB 0GB }
lowerdisacc = sequence { OME 4GB 0GA }
```

While the sequence command is the most direct method to build a linear glycan, it is not the only method. Alternatives that facilitate building more complex glycans and glycoproteins are presented below. For those who need to build structures (and generate topology and coordinate files) that are more complex, a convenient interface that uses GLYCAM is available on the internet (*http://glycam.ccrc.uga.edu* or *http://www.glycam.org*).

Throughout this section, sequences of LEaP commands will be entered in the following format:

```
command argument(s) # descriptive comment
```

This format was chosen so that the lines can be copied directly into a file to be read into LEaP. The number sign (#) signifies a comment. Comments following commands may be left in place for future reference and will be ignored by LEaP. Files may be read into LEaP either by sourcing the file or by specifying it on the command line at the time that LEaP is invoked, e.g.:

```
tleap -f leap_input_file
```

Note that any GLYCAM parameter set shipped with Amber is likely to be updated in the future. The current version is *GLYCAM_06j.dat*. This file and *GLYCAM_06j-1.prep* are automatically loaded with the default *leaprc.GLYCAM_06j-1*. The user is encouraged to check *www.glycam.org* for updated versions of these files.

13.6.1. Procedures for building oligosaccharides using the GLYCAM-06 parameters

13.6.1.1. Example: Linear oligosaccharides

This section contains instructions for building a simple, straight-chain tetrasaccharide:

α-D-Man*p*-(1-3)-β-D-Man*p*-(1-4)-β-D-Glc*p*NAc-(1-4)-β-D-Glc*p*NAc-OH

First, it is necessary to determine the GLYCAM residues that will be used to build it. Since the initial α-D-Man*p* residue links only at its anomeric site, the first character in its name is 0 (zero), indicating that it has no branches or other connections, i.e., it is terminal. Since it is a D-mannose, the second character, the one-letter code, is M (capital). Since it is an α-pyranose, the third character is A. Therefore, the first residue in the sequence above is 0MA. Since the second residue links at its 3-position as well as at the anomeric position, the first character in its name is 3, and, being a β-pyranose, it is 3MB. Similarly, residues three and four are both 4YB. It will also be necessary to add an OH residue at the end to generate a complete molecule. Note that in Section 13.6.3, below, the terminal OH *must* be omitted in order to allow subsequent linking to a protein or lipid. Note also that when present, a terminal OH (or OME etc) is assigned its own residue number.

Converting the order for use with the sequence command in LEaP, gives:

```
Residue name sequence: ROH 4YB 4YB 3MB 0MA
Residue number:         1   2   3   4   5
```

Here is a set of LEaP instructions that will build the sequence (there are, of course, other ways to do this):

```
source leaprc.GLYCAM_06j-1 # load leaprc
glycan = sequence { ROH 4YB 4YB 3MB 0MA } # build oligosaccharide
```

Using the `sequence` command, the ϕ angles are automatically set to the orientation that is expected on the basis of the exo-anomeric effect (± 60°). If you wish to change the torsion angle between two residues, the `impose` command may be used. In the following example, the ψ angles between the two 4YB residues and between the 4YB and the 3MB are being set to the standard value of zero.

```
impose glycan {3} { {C1 O4 C4 H4 0.0} } # set psi between 4YB (3) & 4YB (2)
impose glycan {4} { {C1 O4 C4 H4 0.0} } # set psi between 3MB (4) & 4YB (3)
```

You may now generate coordinate, topology and PDB files, for example:

```
saveamberparm glycan glycan.top glycan.crd # save top & crd
savepdb glycan glycan.pdb # save pdb file
```

13.6.1.2. Example: Branched oligosaccharides

This section contains instructions for building a simple branched oligosaccharide. The example used here builds on the previous one. Again, it will be assumed that the carbohydrate is not destined to be linked to a protein or a lipid. If it were, one should omit the ROH residue from the structure. The branched oligosaccharide is

α-D-Man*p*-(1–3)-β-D-Man*p*-(1–4)-β-D-Glc*p*NAc-(1–4)-β-D-Glc*p*NAc-OH
6
|
α-D-Man*p*-1

Note that the β-D-mannopyranose is now branched at the 3- and 6-positions. Consulting Tables 3.5 to 3.8 informs us that the first character assigned to a carbohydrate linked at the 3- and 6-positions is V. Thus, the name of the residue called 3MB in the previous section must change to VMB.

Thus, when rewritten for LEaP this glycan becomes:

```
Residue name sequence: ROH 4YB 4YB VMB 0MA 0MA
Residue number:         1   2   3   4   5   6
```

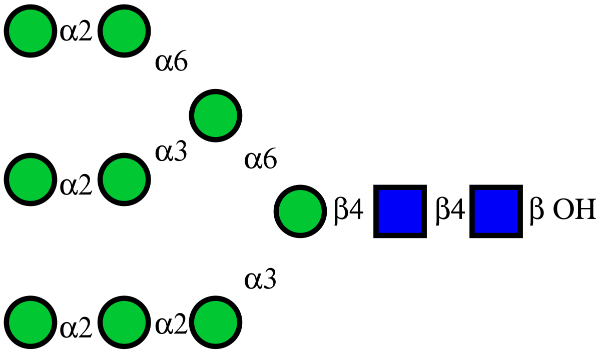


Figure 13.2.: *Structure of Man-9, represented in the symbolic notation used by the Consortium for Functional Glycomics. Here,* 🟢 =D-*Manp and* 🟦 =D-*GlcpNAc*

To ensure that the correct residues are linked at the 3- and 6-positions in VMB, it is safest to specify these linkages explicitly in LEaP. In the current example, the two terminal residues are the same (0MA), but that need not be the case.

```
source leaprc.GLYCAM_06j-1 # load leaprc
glycan = sequence { ROH 4YB 4YB VMB } # linear sequence to branch
```

The longest linear sequence is built first, ending at the branch point "VMB" in order to explicitly specify subsequent linkages. The following commands will place a terminal, 0MA residue at the number three position:

```
set glycan tail glycan.4.O3 # set attachment point to the O3 in VMB
glycan = sequence { glycan 0MA } # add one of the 0MA's
```

The following commands will link the other 0MA to the 6-position. Note that the name of the molecule changes from "glycan" to "branch". This change is not necessary, but makes such command sequences easier to read, particularly with complex structures.

```
set glycan tail glycan.4.O6 # set attachment point to the O6 in VMB
branch = sequence { glycan 0MA } # add the other 0MA
```

It can be especially important to reset torsion angles when building branched oligosaccharides. The following set of commands cleans up the geometry considerably and then generates a set of output files:

```
impose branch {4} { {H1 C1 O6 C6 -60.0} } # set phi torsion and
impose branch {4} { {C1 O6 C6 H6 0.0} } # set psi 0MA(6) & VMB
impose branch {4} { {H1 C1 O4 C4 60.0} } # set phi torsion and
impose branch {4} { {C1 O4 C4 H4 0.0} } # set psi 3MB & 4YB
impose branch {3} { {H1 C1 O4 C4 60.0} } # set phi torsion and
impose branch {3} { {C1 O4 C4 H4 0.0} } # set psi 4YB & 4YB
impose branch {5} { {H1 C1 O3 C3 -60.0} } # set phi torsion and
impose branch {5} { {C1 O3 C3 H3 0.0} } # set psi 0MA(3) & VMB
saveamberparm branch branch.top branch.crd # save top & crd
savepdb branch branch.pdb # save pdb
```

13.6.1.3. Example: Complex branched oligosaccharides

The following example builds a highly branched, high-mannose structure shown in Figure 13.2 . In this example, it is especially important to note that when the branching is ambiguous, LEaP might not choose the attachment point one wants or expects. For this reason, connectivity should be specified explicitly whenever the structure branches. That is, one cannot specify the longest linear sequence and add branches later. The sequence command must be interrupted at each branch point. Otherwise, the connectivity is not assured. In this example, a branch occurs at each VMA (-3,6-D-Man*p*) residue.

The following set of commands, given to tleap, will safely produce the structure represented in Figure13.2.

```
source leaprc.GLYCAM_06j-1
glycan = sequence { ROH 4YB 4YB VMB }
set glycan tail glycan.4.O6
glycan=sequence { glycan VMA }
set glycan tail glycan.5.O6
glycan=sequence { glycan 2MA 0MA }
set glycan tail glycan.5.O3
glycan=sequence { glycan 2MA 0MA }
set glycan tail glycan.4.O3
glycan=sequence { glycan 2MA 2MA 0MA }
impose glycan {3} { {H1 C1 O4 C4 60.0} }
impose glycan {3} { {C1 O4 C4 H4 0.0} }
impose glycan {4} { {H1 C1 O4 C4 60.0} }
impose glycan {4} { {C1 O4 C4 H4 0.0} }
impose glycan {5} { {H1 C1 O6 C6 -60.0} } # 1-6 Link from (5) to (4), Phi
impose glycan {5} { {C1 O6 C6 C5 180.0} } # 1-6 Link from (5) to (4), Psi
impose glycan {4} { {O6 C6 C5 O5 60.0} } # 1-6 Link from (5) to (4), Chi
impose glycan {10} { {H1 C1 O3 C3 -60.0} }
impose glycan {10} { {C1 O3 C3 H3 0.0} }
impose glycan {6} { {H1 C1 O6 C6 -60.0} }
impose glycan {6} { {C1 O6 C6 C5 180.0} }
impose glycan {5} { {O6 C6 C5 O5 -60.0} }
impose glycan {8} { {H1 C1 O3 C3 -60.0} }
impose glycan {8} { {C1 O3 C3 H3 0.0} }
impose glycan {7} { {H1 C1 O2 C2 -60.0} }
impose glycan {7} { {C1 O2 C2 H2 0.0} }
impose glycan {9} { {H1 C1 O2 C2 -60.0} }
impose glycan {9} { {C1 O2 C2 H2 0.0} }
impose glycan {11} { {H1 C1 O2 C2 -60.0} }
impose glycan {11} { {C1 O2 C2 H2 0.0} }
impose glycan {12} { {H1 C1 O2 C2 -60.0} }
impose glycan {12} { {C1 O2 C2 H2 0.0} }
saveamberparm glycan glycan.prmtop glycan.restrt
```

13.6.2. Procedures for building a lipid using GLYCAM-06 parameters

The procedure described here allows a user to produce a single lipid molecule without consideration for axial alignment. Lipid bilayers are typically built in the (x,y) plane of a Cartesian coordinate system, which requires the individual lipids to be aligned hydrophilic “head” to hydrophobic “tail” along the z-axis. This can be done relatively easily by loading a template PDB file that has been appropriately aligned on the z-axis.

The lipid described in this example is 1,2-dimyristoyl-*sn*-glycero-3-phosphocholine or DMPC. For this example, DMPC will be composed of four fragments: CHO, the choline “head” group; PGL, the phospho-glycerol “head” group; MYR, the sn-1 chain myristic acid “tail” group; and MY2, the sn-2 chain myristic acid “tail” group. See the molecular diagram in13.3 for atom labels (hydrogens and atomic charges are removed for clarity) and bonding points between each residue (dashed lines). This tutorial will use only prep files for each of the four fragments. These prep files were initially built as PDB files and formatted as prep files using *antechamber*. GLYCAM-compatible charges were added to the prep files and a prep file database (GLYCAM_lipids_06h.prep) was created containing all four files.

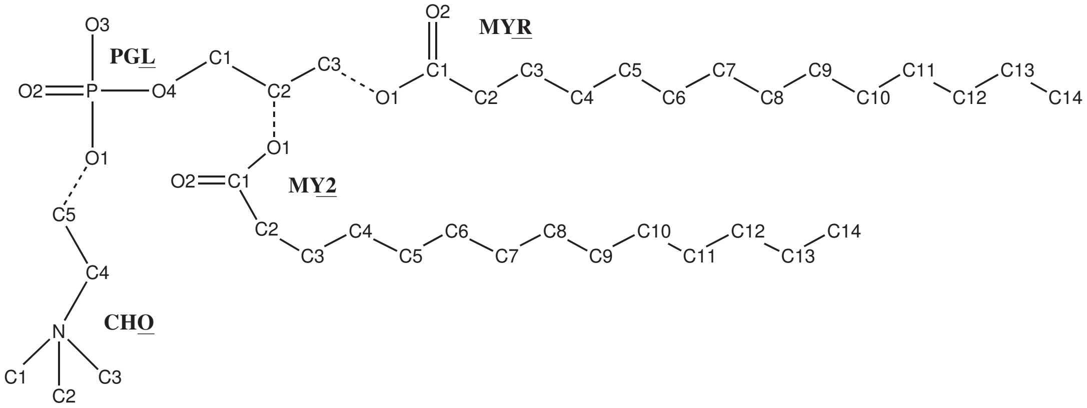


Figure 13.3.: *DMPC*

13.6.2.1. Example: Building a lipid with LEaP.

One need not load the main GLYCAM prep files in order to build a lipid using the GLYCAM-06 parameter set, but it is automatically loaded with the default *leaprc.GLYCAM_06j-1*. Note that the lipid generated by this set of commands is not necessarily aligned appropriately to create a bilayer along an axis. The commands to use are:

```
source leaprc.GLYCAM_06j-1 # source the leaprc for GLYCAM-06
loadamberprep GLYCAM_06_lipids.prep # load the lipid prep file
set CHO tail CHO.1.C5 # set the tail atom of CHO as C5.
set PGL head PGL.1.O1 # set the head atom of PGL to O1
set PGL tail PGL.1.C3 # set the tail atom of PGL to C3
lipid = sequence { CHO PGL MYR } # generate the straight-chain
# portion of the lipid
set lipid tail lipid.2.C2 # set the tail atom of PGL to C2
lipid = sequence { lipid MY2 } # add MY2 to the "lipid" unit
impose lipid {2} { {C1 C2 C3 O1 163} } # set torsions for
impose lipid {2} { {C2 C3 O1 C1 -180} } # PGL & MYR
impose lipid {2} { {C3 O1 C1 C2 180} }
impose lipid {2} { {O4 C1 C2 O1 -60} } # set torsions for
impose lipid {2} { {C1 C2 O1 C1 -180} } # PGL & MY2
impose lipid {2} { {C2 O1 C1 C2 180} }
# Note that the values here may not necessarily
# reflect the best choice of torsions.
savepdb lipid DMPC.pdb # save pdb file
saveamberparm lipid DMPC.top DMPC.crd # save top and crd files
```

13.6.3. Procedures for building a glycoprotein in LEaP.

The LEaP commands given in this section assume that you already have a PDB file containing a glycan and a protein in an appropriate relative configuration. Thorough knowledge of the commands in LEaP is required in order to successfully link any but the simplest glycans to the simplest proteins, and is beyond the scope of this discussion. Several options for generating the relevant PDB file are given below (see Items 5a-5c).

The protein employed in this example is bovine ribonuclease A (PDBID: 3RN3). Here the branched oligosaccharide assembled in the second example will be attached (*N*-linked) to ASN 34 to generate ribonuclease B.

13.6.3.1. Setting up protein pdb files for glycosylation in LEaP.

1. Delete any atoms with the "HETATM" card from the PDB file. These would typically include bound ligands, non-crystallographic water molecules and non-coordinating metal ions. Delete any hydrogen atoms if

present.

2. In general, check the protein to make sure there are no duplicate atoms in the file. This can be quickly done by loading the protein in LEaP and checking for such warnings. In this particular example, residue 119 (HIS) contained duplicate side chain atoms. Delete all but one set of duplicate atoms.

3. Check for the presence of disulfide bonds (SSBOND) by looking at the header section of the PDB file. 3RN3 has four disulfide bonds, between the following pairs of cysteine residues: 26—84, 40—95, 58—110, and 65—72. Change the names of these eight cysteine residues from CYS to CYX.

4. At present, it is possible to link glycans to serine, threonine, hydroxyproline and asparagine. You must rename the amino acid in the protein PDB file manually prior to loading it into LEaP. The modified residue names are OLS (for *O*-linkages to SER), OLT (for *O*-linkages to THR), OLP (for *O*-linkages to hydroxyproline, HYP) and NLN (for *N*-linkages to ASN). Libraries containing amino acid residues that have been modified for the purpose are automatically loaded when *leaprc.GLYCAM_06j-1* is sourced. See the lists of library files in3.3 for more information.

5. Prepare a PDB file containing the protein and the glycan, with the glycan correctly aligned relative to the protein surface. There are several approaches to performing this including:

    a) It is often the case that one or more glycan residues are present in the experimental PDB file. In this case, a reasonable method is to superimpose the linking sugar residue in the GLYCAM-generated glycan upon that present in the experimental PDB file, and to then save the altered coordinates. If you use this method, remember to delete the experimental glycan from the PDB file! It is also essential to ensure that each carbohydrate residue is separated from other residues by a TER card in the PDB file. Also remember to delete the terminal OH or OMe from the glycan. Alternately, the experimental glycan may be retained in the PDB file, provided that it is renamed according to the GLYCAM 3-letter code, and that the atom names and order in the PDB file match the GLYCAM standard. This is tedious, but will work. Again, be sure to insert TER cards if they are missing between the protein and the carbohydrate and between the carbohydrate residues themselves.

    b) Use a molecular modeling package to align the GLYCAM-generated glycan with the protein and save the coordinates in a single file. Remember to delete the terminal OH or OMe from the glycan.

    c) Use the Glycoprotein Builder tool at *http://www.glycam.org*. This tool allows the user to upload protein coordinates, build a glycan (or select it from a library), and attach it to the protein. All necessary AMBER files may then be downloaded. This site is also convenient for preprocessing protein-only files for subsequent uploading to the glycoprotein builder.

13.6.3.2. Example: Adding a branched glycan to 3RN3 (*N*-linked glycosylation).

In this example we will assume that the glycan generated above ("branch.pdb") has been aligned relative to the ASN 34 in the protein file and that the complex has been saved as a new PDB file (e.g., as "3rn3_nlink.pdb"). The last amino acid residue should be VAL 124, and the glycan should be present as 4YB 125, 4YB 126, VMB 127, OMA 128 and OMA 129.

Remember to change the name of ASN 34 from ASN to NLN. For the glycan structure, ensure that each residue in the PDB file is separated by a "TER" card. *The sequence command is not to be used here, and all linkages (within the glycan and to the protein) will be specified individually.*

Enter the following commands into *xleap* (or *tleap* if a graphical representation is not desired). Alternately, copy the commands into a file to be sourced.

```
source leaprc.GLYCAM_06j-1 # load the GLYCAM-06 leaprc for ff14SB
source leaprc.protein.ff14SB # load the protein force field
glyprot = loadpdb 3rn3_nlink.pdb # load protein and glycan pdb file
bond glyprot.125.O4 glyprot.126.C1 # make inter glycan bonds
bond glyprot.126.O4 glyprot.127.C1
bond glyprot.127.O6 glyprot.128.C1
bond glyprot.127.O3 glyprot.129.C1
bond glyprot.34.SG glyprot.125.C1 # make glycan -- protein bond
bond glyprot.26.SG glyprot.84.SG # make disulfide bonds
bond glyprot.40.SG glyprot.95.SG
bond glyprot.58.SG glyprot.110.SG
bond glyprot.65.SG glyprot.72.SG
addions glyprot CL 0 # neutralize appropriately
solvateBox glyprot TIP3P BOX 8 # solvate the solute
savepdb glyprot 3nr3_glycan.pdb # save pdb file
saveamberparm glyprot 3nr3_glycan.top 3nr3_glycan.crd # save top, crd
quit # exit leap
```

13.6.4. Solvating a system with a specific number of molecules

Sometimes it is desirable to solvate a system with a target number of waters rather than specifying a particular box size. The following script is a wrapper around LEaP which can be used for this purpose:

```
$AMBERHOME/AmberTools/src/etc/Solvate.sh
```

In addition to LEaP (really tleap), the script also makes use of cpptraj for determining molecule info.

13.6.4.1. Solvate.sh Usage

```
Solvate.sh <input_file>
Input File Options: (default)
```

target <#> `Target # of waters to add.`

buffer <buf> `Initial buffer size (10.0).`

bufx <buf> `Initial buffer X size (mode 2|3 only, 10.0).`

bufy <buf> `Initial buffer Y size (mode 2|3 only, 10.0).`

pdb <file> `Solute PDB file name.`

top <name> `Output topology (solvated.parm7).`

crd <name> `Output coordinates (solvated.rst7).`

leapin <file> `Leap input script for loading parameters etc.`

ionsin <file> `Optional Leap input for loading ions etc (run after solvating).`

templeap <name> `Name of temporary leap input script (temp.leap.in).`

tol <#> `Number of waters > target allowed, will be removed (2).`

mode <#> `Solvate mode: (0)- SolvateOct 1 - SolvateBox 2 - SolvateBoxXYZ (bufx and bufy are scaled) 3 - SolvateBoxZ (bufx and bufy are fixed)`

loadpdb {yes|no} `If (yes), use 'loadpdb PDB'; otherwise <leapin> should set up unit <molname>.`

loadcmd <cmd> `Command to load solute file; default 'loadpdb'.`

soluteres <#> `Number of solute residues. If blank try to guess from PDB.`

molname <name> `Solute molecule unit name ('m').`

solventunit <name> `Solvent unit (TIP3PBOX). Recognized solvent units: TIP3PBOX SPCBOX OPCBOX TIP4PEWBOX`

First the file specified by leapin is read by LEaP, then the system is solvated, then the file specified by 'ionsin' is read in order to add ions etc.

13.6.4.2. Solvate.sh Example

Solvate an RNA tetranucleotide with 2500 TIP3P waters and 3 Na+ ions.
Input file: solvate.in

```
# Target number of waters
target 2500
# Initial guess for buffer
buffer 10
# Input PDB name
pdb rGACC.pdb
# Output topology name
top rGACC.tip3p.parm7
# Output coordinates name
crd rGACC.nomin.rst7
# Base leap input script
leapin leap.solvate.in
# Additional script for adding ions etc
ionsin leap.ions.in
# Tolerance (# of waters off from target allowed)
tol 3
# 0 - SolvateOct
mode 0
```

LEaP input: leap.solvate.in

```
source leaprc.RNA.OL3
set default pbradii mbondi2
```

LEaP Ions input: leap.ions.in

```
addions m Na+ 1
addions m Na+ 1
addions m Na+ 1
```

14. Reading and modifying Amber parameter files

This chapter describes the content of Amber parameter files, along with details about *ParmEd* (which can be used to examine and modify prmtop files) and *paramfit* (which can be used to fit force fields to quantum mechanical and other target data).

14.1. Understanding Amber parameter files

Romain M. Wolf, Jason Swails, and David A. Case

This chapter provides a short description of Amber-compatible force field parameter files is given. Only the actual data in parameter (**.dat*) files are discussed. The special issue of deriving partial charges is not addressed. Also, more complex subjects dealing with parameters for implicit solvent (GB or PB) or polarisability computations are skipped. This text is meant as a documentation for users who want to understand parameter files, and in some cases might be tempted to change or add some parameters. Most of the following documentation is found in bits and pieces at various Amber-related sites and in tutorials or original Amber manuals and these various sources have been helpful to put together this hopefully concise documentation.

14.1.1. Parameter Transfers between Force Fields

Transferring parameters from one force field to another must respect the underlying functional form, the units in which parameters are expressed in the parameter files, and also the exact procedures on how individual parameters were obtained. In addition, attention must be paid to the methods used to deduce partial charges. Force fields are self-consistent, i.e., all terms are interrelated and their actual values depend on the way they were derived. Therefore, any parameter transfer between different force fields is dangerous, even when the functional form is the same (or looks as if it were...).

Torsion terms are the most critical. Many torsion barriers and profiles are not easily assessed experimentally and are often deduced from *ab initio* quantum mechanical (QM) computations on small fragments. Since QM calculations offer many possibilities, the exact nature of these calculations (basis sets, Hartree-Fock and/or density functionals, etc.) used to derive parameters should be known.

Special care must also be applied to 1-4 interactions, i.e., interactions between atoms separated by exactly three consecutive bonds. Most Amber force fields for example assume that 1-4 interactions get a special treatment. See section 14.1.6 for details. In many other force fields, the special treatment of 1-4 interactions is either different or non-existent. This has an immediate influence on the torsion terms and resulting conformation energies. Therefore, before transferring torsion terms, van der Waals parameters and partial charges from other force fields, check the special treatment of 1-4 interactions in the source and the target force field.

14.1.2. How Amber Routines Use the Parameter Files

Amber routines that perform actual calculations (sander, pmemd, etc...) do not read parameter files directly. They use a special file type, the ***parameter-topology*** file (*parmtop* from now on), which contains all the information required by the various energy functions in the computation routines. The *parmtop* file is specific to the molecular system for which it was created and is directly related to the second required file, the coordinate file.[1] Smallest changes to the system (adding or removing atoms, or even changing the order of atoms in the coordinate file) render the *parmtop* useless.

---

[1] This file can be in the Amber coordinate ’crd’ file format or, for some applications, also in PDB format.

Although *parmtop* files are pure ASCII files, changing parameters directly in them by standard text editors is strongly discouraged. In the worst case, computations will run without any warnings, but results might be totally flawed. The safest way to generate *parmtop* files is to use an Amber tool like *tleap* that has been used, tested, and enhanced over a number of years and usually generates correct *parmtop* files, provided that the input is correct and that all required information is available via fragment libraries and parameter files. The latest AmberTools 12.0 version (April 2012) includes the *ParmEd* python script of Jason Swails which is very useful to examine or post-process *parmtop* files. However, only users with detailed knowledge on the exact format of *parmtop* files should dare fiddling around with this data type.

14.1.3. "*.dat" and "frcmod.*" Files

The standard parameter files with the `.dat` extension are located in the folder `$AMBERHOME/dat/leap/parm`. Adding or changing parameters directly in the parameter files delivered with an Amber distribution is not a good idea for the following reasons: *(a)* you might mess up the parameter file, *(b)* you might have trouble to remember and find your changes later and add confusion when publishing results, *(c)* subsequent updates or patches might overwrite your changes.

In the above mentioned folder, there are also various `frcmod.*` files. They have basically the same format as the parameter `*.dat` files. See some of the examples provided in the Amber distributions. These files can be read into *tleap* exactly like the standard `*.dat` files. They merge the default parameters in the `*.dat` file with the new parameters in the `frcmod.*` files. More important, if the same parameters already exist in the `*.dat` files, the parameters in the `frcmod.*` files overwrite the default `*.dat` parameters. This offers a handy way to add new or to change original parameters without ever touching the default parameter files. Just make sure to read the respective `frcmod.*` files in *tleap* when the new or altered parameters should be used.

14.1.4. Parameters Required for Amber Force Fields

The simplest form of the Amber force field (neglecting implicit solvent or polarisation terms) uses the following Hamiltonian:

$$\begin{aligned} E_{total} &= \sum_{bonds} k_b(r-r_0)^2 \\ &+ \sum_{angles} k_\theta(\theta-\theta_0)^2 \\ &+ \sum_{dihedrals} V_n[1+\cos(n\phi-\gamma)] \\ &+ \sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\left[\frac{A_{ij}}{R_{ij}^{12}}-\frac{B_{ij}}{R_{ij}^{6}}+\frac{q_iq_j}{\varepsilon R_{ij}}\right] \end{aligned} \tag{14.1}$$

In this equation, the terms $k_b, r_0, k_\theta, \theta_0, V_n, \gamma, A_{ij}, B_{ij}$ are parameters to be specified in the parameter files mentioned in section 14.1.3 for the various Amber force fields.[2] The meaning of these different parameters is outlined in the following sections.

Equation 14.1 does not have a special term for out-of-plane motions. Amber routines handle these terms through the same formulation as the torsion terms (see section 14.1.6).

Partial charges (q_i, q_j in equation 14.1), although parameters also, do not appear in parameter files, but are assigned differently (see 14.1.7).

---

[2] Note that equation 14.1 does not use the (physically more correct) $\frac{k_b}{2}$, $\frac{k_\theta}{2}$, and $\frac{V_n}{2}$ notations because it refers to the constants as they appear in the actual parameter files.

14.1.5. Atom Types

Amber atom types can be one or two characters long. Uppercase (standard protein and nucleotide force fields), lowercase (GAFF General Amber Force Field), and mixed upper-lowercase (GLYCAM sugar force field) are allowed. Obviously, atom types must have a single, unique, definition.

If considering the definition a new atom type, think about the consequences. Of course, an atom type with an identical name must **not** already exist in one of the standard force fields used in the Amber community. Depending on how often and in how many combinations the atom type might occur, be also aware of the rather large number of additional parameters that might be required. Especially for bond angles, this number can grow very rapidly.

A new atom type definition, if required, must be clear and precise. It should also be possible to treat the definition in an automatic atom-type assignment procedure. Requiring users to visually verify and to change atom types by hand will cause trouble and will make it impossible to use the force field in automatic procedures that should not require user intervention for this task.

14.1.6. Bonded Interaction Terms

Bond Stretching Terms

The first row in equation 14.1 (page 243) is the harmonic term for bond stretching. In Amber-type parameter files, the force constant k_b is given for energy values in kcal/mol, with bond lengths in Å. The following line shows an example from the GAFF force field file `gaff.dat`.

The bond between a sp^3 carbon (`c3`) and a hydroxyl oxygen (`oh`) has a default (equilibrium) value of 1.426 Å and a force constant of 314.1 kcal/mol/Å^2.

```
c3-oh 314.1 1.4260 SOURCE1 914 0.0129
```

The entrance in the parameter file starts with the definition of the bond (`atomtype1 hyphen atomtype2`), followed by the force constant k_b (in kcal/mol/Å^2) and the equilibrium bond length r_0 (in Å). Only the first three fields are relevant for computations. The other fields on the line above are mainly documentation.

As stated before, atom types in Amber FFs cannot have more than two characters. But if they have only one character (e.g., a carbonyl carbon atom `c`), entries with a one-letter atom type must look like this:

```
c -oh 466.4 1.3060 SOURCE1 271 0.0041
```

i.e., the space is **after** the atom type, **before** the hyphen.

Starting with a space like on the next line might lead to problems.

```
 c-oh 466.4 1.3060 SOURCE1 271 0.0041
```

This holds for all parameter file entries that use hyphens to separate atom types, i.e., also angle and torsion terms (see following sections).

Angle Bending Terms

Angle bending terms are parameterised by a force constant k_θ in kcal/mol/radian^2 and an equilibrium angle value θ_0 in degrees. They have the format as shown below:

```
c3-c3-oh 67.720 109.430 SOURCE3 48 1.5023
```

The middle atom `c3` is bonded to another `c3` and to a hydroxyl oxygen `oh`. The equilibrium bond angle θ_0 is 109.43 degrees and the force constant is 67.720 kcal/mol/radian^2. Note that internally, angle deviations are computed in π-radian^2. The *parmtop* files also express the default 'equilibrium' bond angles in radians. For example, the angle of 109.43 degrees is internally represented as 1.9099 π-radians. Using degrees in the original parameter files is obviously more convenient. Anything after the third field, the equilibrium angle, is mainly documentation and not required.

Torsion Terms

The third row in equation 14.1 is the usual Fourier-series expansion for torsional terms. In Amber parameter files, these entries require a careful explanation:

First, many torsion terms contain generic entries, using the notation '`X`' for 'any atom'. These terms are used when the parameter file does not contain more specific terms for the same torsion. They are combined with explicit terms when present. Entries with generic '`X`' atoms must always come **before** the more specific ones in the parameter files.

Second, Amber parameter files use a special notation for torsions that require more than one torsional term (see example towards the end of section 14.1.6).

Third, the parameter file entry not only contains the torsion barrier term V_n (in kcal/mol), the phase γ (degrees) and the periodicity n, but also a **divider** (integer) which splits the torsion term into individual contributions for each pair of atoms involved in the torsion.

Fourth, torsion entries can also contain information about the special scaling of 1-4 non-bonded interactions (see section 14.1.6 on page 247).

Consider the following example, the default term for the torsion around a C_{sp3}-C_{sp3} single bond:

```
X -c3-c3-X 9 1.400 0.000 3.000 JCC,7,(1986),230
```

The five relevant terms on this line are:

1. the definition (`X -c3-c3-X`)
2. the divider (`9`)
3. the barrier term (`1.400`)
4. the phase (`0.000`)
5. the periodicity (`3.000`)

Fields after the periodicity are mainly comment, **except for the special flags `SCNB` and `SCEE`**, that, if present, govern the special treatment of 1-4 non-bonded interaction (see section 14.1.6)

The torsional barrier term (the actual barrier divided by two) is 1.400 and the periodicity is 3. The **phase** is **zero** in this example, meaning that a **maximum** energy is encountered at zero degrees. A **phase of 180 degrees** on the other hand means that there is a **minimum** at 180 degrees. The divider is 9 because each C_{sp3} has three X attached to it and each X 'sees' three X attached to the other C_{sp3} ($3 \times 3 = 9$).

For a torsion angle ϕ (defined as X-c3-c3-X) of -60, 60, or 180 degrees, the torsion energy term would be zero:

$$\frac{1.4}{9} \times [1 + \cos(3 \times \phi - 0.0)] = 0 \tag{14.2}$$

This corresponds to the staggered conformation, i.e., the lowest energy state in a X_3C-CX_3 connectivity like for example ethane (H_3C-CH_3)

By rotating around the C-C bond, an eclipsed conformation where the X are exactly opposed is encountered three times (periodicity = 3), namely at ϕ = 0, 120, or 240 (-120) degrees.

$$\frac{1.4}{9} \times [1 + \cos(3 \times \phi - 0.0)] = 0.3111 \tag{14.3}$$

.

Since the divider is 9, we have to multiply the value of 0.3111 by 9 to get the full torsional barrier, i.e., $9 \times 0.3111 = 2.8$ kcal/mol.[3] This might be used for ethane for example and would be close to the experimental torsion barrier (ca. 3 kcal/mol).

In GAFF however, there is also a specific term for `hc-c3-c3-hc` that would come into play for ethane. In this case, the divider is 1, because the term is fully defined.

---

[3] The actual barrier value of 2.8 kcal/mol here is twice the barrier term of 1.4 in the parameter file.

```
hc-c3-c3-hc 1 0.15 0.0 3. Junmei et al, 1999
```

Thus, using GAFF for ethane, this term counts 9 times because there are nine [hc,hc] pairs seeing each other. Instead of equation 14.3, one would use

$$0.15 \times [1 + \cos(3 \times \phi - 0.0)] = 0.3000 \tag{14.4}$$

i.e., the total torsional term in ethane would be $9 \times 0.3 = 2.7$ kcal/mol. The experimental torsional barrier value of ca. 3 kcal/mol would be reached because of the additional van der Waals and Coulomb repulsion terms between the staggered hydrogens.

Assume a connectivity for which some terms are fully defined (all four atom types are specified) while no specific entry is given for others. In that case, the equations are combined. The specific terms are counted once (divider = 1) and the remaining general terms are added according to

$$\frac{V_{barrier}}{divider} \times [1 + \cos(periodicity \times \phi - phase)] \tag{14.5}$$

Things get more complex when the Fourier series has more than one term. A typical example would be the rotation around an amide bond R1-NH-C(=O)-R2. In this case, the trans amide (H and O on opposite sides, $\phi = 180°$) is preferred over the cis-amide (H and O on the same side, $\phi = 0$). The entry in the GAFF parameter file for this torsion is

```
hn-n -c -o 1 2.50 180.0 -2. JCC,7,(1986),230
hn-n -c -o 1 2.00   0.0  1. J.C.cistrans-NMA
```

If the torsion definition has a "negative" periodicity (-2 in the case above), it tells programs reading the parameter file that additional terms are present for that particular connectivity. The equation to be applied for `hn-n -c -o` is:

$$E_{torsion} = 2.00 \times [1 + \cos(1 \times \phi - 0.0)] + 2.50 \times [1 + \cos(2 \times \phi - 180.0)] \tag{14.6}$$

Equation 14.6 prefers the *trans* amide ($\phi = 180°$) over the *cis* amide ($\phi = 0$) by 4 kcal/mol considering the torsion term alone. However the more favourable Coulomb term (the 1-4 attractive interaction between the negative carbonyl oxygen and the positive amide hydrogen) reduces the overall preference for the *trans* conformation close to the experimental value of ca. 2 kcal/mol.

In addition, the following general terms have to be applied for the torsions involving R1 and R2 in the peptide bond R1-NH-C(=O)-R2, in order to compute the high torsional barrier of an amide bond:

```
X -c -n -X 4 10.000 180.000 2.000
```

Torsional terms are obviously the most difficult part to parametrize in a force field. They are in a way the last rescue to get torsional barriers right, after all other terms have been adjusted. Therefore, their transfer from one force field to the other is always most risky and acceptable only if all other involved terms in two force fields are very similar. Transferability must always be validated.

Out-of-Plane Terms

Out-of-plane terms are handled via a Fourier term, similar to the torsion terms. But the four involved atoms are not serially (linearly) bonded, they are "branched". The "central" atom is the atom that is forced into the plane of the other three. For example, to keep a carbonyl group R1-C(=O)-R2 planar, the central C atom must be forced into the plane of the other three connected items R1, R2, and O. The entry in the GAFF parameter file for this term is

```
X -X -c -o 10.5 180. 2. JCC,7,(1986),230
```

Note that in Amber the central atom type (here `c`) is the **third** in the definition.The order of the remaining atoms should (by definition) be alphabetic in atom type. The phase is always 180°. In all-atom force fields, the periodicity is always 2.

Out-of-plane terms are the only terms that are allowed to be "missing" in Amber parameter files. Common ones are added automatically by tools like *tleap*. In many cases, these terms are "cosmetics" that avoid "in principle" planar structures from getting distorted under the influence of other forces (e.g., fused rings, planar nitrogens with three substituents, etc...). The actual parameterisation is often intuitive and for many entries, the ("generic") parameters are identical.

1-4 Non-Bonded Interaction Scaling

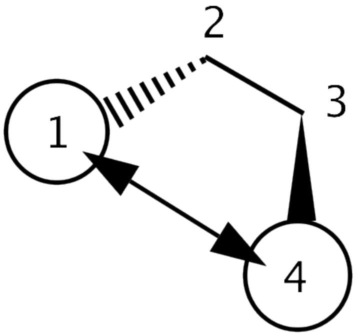


Figure 14.1.: *1-4 Interactions between atoms "1" and "4".*

Non-bonded interactions between atoms separated by three consecutive bonds (as schematically shown in Figure 14.1) require a special treatment in Amber force fields. Although referring to non-bonded interactions, scaling information is included in the torsion terms part of the parameter files.

By default, vdW 1-4 interactions are divided (scaled down) by a factor of 2.0, electrostatic 1-4 terms by a factor of 1.2. These are default values for the protein force fields and GAFF, but not for sugar force fields GLYCAM_06EP and GLYCAM_06, for example, in which these interactions are not scaled at all.

Without any additional information, programs like *tleap*, used to prepare *parmtop* files, assume that the standard scaling mentioned above is to be applied. However, this default can be overwritten in the torsion section of the parameter file. An example is shown below for torsional terms in the GLYCAM_06j force field:

```
S -Ng-Cg-H1 1 2.00 0.0 1. SCEE=1.0 SCNB=1.0 N-Sulfates
S -Ng-Cg-Cg 1 0.0 0.0 -3. SCEE=1.0 SCNB=1.0 N-Sulfates
```

The special notation `SCEE=1.0 SCNB=1.0` following the standard torsion terms[4] will tell *tleap* to prepare a *parmtop* file which transfers these data into a special section, as shown below:

```
%FLAG SCEE_SCALE_FACTOR
%FORMAT(5E16.8)
scaling factors are entered here....
%FLAG SCNB_SCALE_FACTOR
%FORMAT(5E16.8)
scaling factors are entered here....
```

When using standard Amber force field parameter files as delivered with AmberTools, the user does not need to care about this. However, when adding additional parameters, especially torsion terms, one should be aware of these scaling factors and decide if they should be default or altered.

14.1.7. Non-Bonded Terms

Van der Waals Parameters

The standard formulation of the 6-12 Lennard-Jones potential $V_{i,j}$ between two atoms i and j is:

---

[4] In this case, the fields coming after the periodicity (field 5), i.e., fields 6 and 7 are also read and are not 'just' comment!

$$V_{i,j} = 4\varepsilon_{i,j}\left[\left(\frac{\sigma_{i,j}}{r_{i,j}}\right)^{12} - \left(\frac{\sigma_{i,j}}{r_{i,j}}\right)^{6}\right] \tag{14.7}$$

Here, $r_{i,j}$ is the distance separating the two atoms, $\varepsilon_{i,j}$ is the depth of the potential well for the interaction of atoms i and j, and $\sigma_{i,j}$ is the distance where the potential is exactly zero, i.e., where 'repulsion' starts for the two atoms. Both $\varepsilon_{i,j}$ and $\sigma_{i,j}$ are specific for **the pair** of atoms (or more precisely, 'atom types').

Another possible formulation of $V_{i,j}$, relating to the concept of van der Waals radii, is:

$$V_{i,j} = \varepsilon_{i,j}\left[\left(\frac{R_{min}}{r_{i,j}}\right)^{12} - 2\left(\frac{R_{min}}{r_{i,j}}\right)^{6}\right] \tag{14.8}$$

In this case, R_{min} is the sum of the van der Waals radii, $R_i + R_j$ of atoms i and j, the contact distance at which the potential is at its minimum, i.e., at a value of $-\varepsilon$.

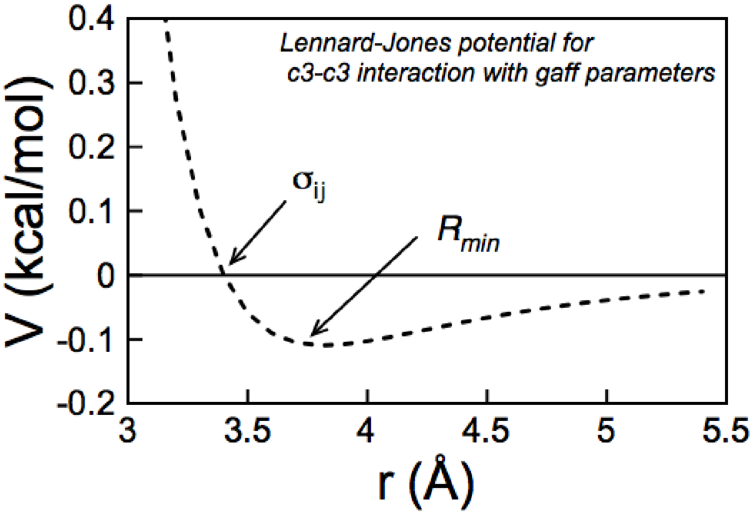


Figure 14.2.: *Example of Lennard-Jones potential: the used data are those for the c3 atom type in the* gaff *force field (vdW radius R_{min}= 1.908 Å, ε = 0.1094 kcal/mol)*

Combining equations (14.7) and (14.8) gives for the relation between σ and R_{min}:

$$R_{min} = 2^{1/6}\sigma \;\; or \;\; \sigma = 2^{-1/6}R_{min} \tag{14.9}$$

In force fields, the 'A,B' notation of the Lennard-Jones potential is commonly used:

$$V_{i,j} = \frac{A_{i,j}}{r_{i,j}^{12}} - \frac{B_{i,j}}{r_{i,j}^{6}} \tag{14.10}$$

where $A_{i,j}$ and $B_{i,j}$ are specific parameters for atom type pairs i and j. The meaning of $A_{i,j}$ and $B_{i,j}$ are easily deduced from equation (14.7):

$$A = 4\varepsilon\sigma^{12} \;\; and \;\; B = 4\varepsilon\sigma^{6} \tag{14.11}$$

or, in terms of R_{min}, using equation (14.8):

$$A = \varepsilon R_{min}^{12} \;\; and \;\; B = 2\varepsilon R_{min}^{6} \tag{14.12}$$

Van der Waals data in Amber force fields are given for each atom as a single data pair, a radius R_{min} ('van der Waals' radius in Å) and an energy ε (kcal/mol) representing the depth of the potential well.

These values are given at the end of the force field parameter files. In protein force fields, lines above these data show equivalences. For example the line

```
N NA N2 N* NC NB NT NY
```

indicates that all atom types following N (the amide nitrogen) inherit the same Lennard-Jones parameters. Thus, no entry for NA, N2, ... has to be given explicitly.

For Amber force fields, cross terms involving different atom types i and j are evaluated according to the Lorentz/Berthelot mixing rules:

$$\sigma_{i,j} = 0.5(\sigma_{i,i} + \sigma_{j,j}) \;\; or \;\; R_{min,i,j} = 0.5(R_{min,i} + R_{min,j}) \tag{14.13}$$

$$\varepsilon_{i,j} = \sqrt{\varepsilon_{i,i} \cdot \varepsilon_{j,j}} \tag{14.14}$$

The *parmtop* file entries are in 'A' and 'B' terms to be used directly with equation 14.10, transforming the $[R_{min},\varepsilon]$ data pairs from the parameter files.

As an example, consider ethanol (CH_3CH_2OH) with the GAFF force field. There are five different GAFF atom types. Below are shown the corresponding $[R_{min},\varepsilon]$ data pairs, as found in the *gaff.dat* parameter file:

```
h1 1.3870 0.0157 Veenstra et al JCC,8,(1992),963
hc 1.4870 0.0157 OPLS
ho 0.0000 0.0000 OPLS Jorgensen, JACS,110,(1988),1657
oh 1.7210 0.2104 OPLS c3 1.9080 0.1094 OPLS
```

Note that there are three different hydrogen types: `hc`, the default H atom connected to an aliphatic carbon, `h1`, a hydrogen type connected to an aliphatic carbon with one electronegative substituent (the oxygen in this case), and the hydroxyl hydrogen `ho` (for which van der Waals interactions are neglected in Amber).

Partial Charges

For Amber force fields, partial charges do not appear in parameter files. For proteins and nucleic acid force fields that use fragment (residue) libraries, partial charges are pre-defined and have been computed from electrostatic-potential fitting of high-level *an initio* QM. They are automatically assigned by tools like *tleap*. Library files are found the folder `$AMBERHOME/dat/leap/lib`.

Below is shown the alanine (ALA) residue of the library file `all_amino94.lib`:

```
"N" "N" 0 1 131072 1 7 -0.415700
"H" "H" 0 1 131072 2 1 0.271900
"CA" "CT" 0 1 131072 3 6 0.033700
"HA" "H1" 0 1 131072 4 1 0.082300
"CB" "CT" 0 1 131072 5 6 -0.182500
"HB1" "HC" 0 1 131072 6 1 0.060300
"HB2" "HC" 0 1 131072 7 1 0.060300
"HB3" "HC" 0 1 131072 8 1 0.060300
"C" "C" 0 1 131072 9 6 0.597300
"O" "O" 0 1 131072 10 8 -0.567900
```

The partial charges for each atom are given in the last field of each line.

For the GAFF force field, there are various options to compute partial charges, the AM1-BBC method being probably the best trade-off between quality and speed. There are other file types that can contain user-specified partial charges, e.g., SYBYL mol2 files. See the *antechamber* documentation for details.

In *parmtop* files, partial charges are not entered as fragments of the electron charge, but are multiplied by the square-root of 332.05 (= 18.22), because the factor 332.05 converts the Coulomb energy into kcal/mol when using fragments of the electron charge in the Coulomb term of equation 14.1.

14.1.8. Final Remarks

Most parameters in Amber force fields have been tested on a large variety of structures. In rare cases, situations are encountered where structures look "strange" or where results are obviously wrong. One should first look into details of the simulation conditions and settings before blaming the problem on actually flawed force field parameters. Simple test cases are often helpful to resolve the enigma.

When changing or adding parameters and later publishing results, new parameter should be mentioned. Also, the Amber developers team should be notified about possibly problematic parameters. This ensures that potential errors are corrected via patches in later versions and it will help the entire user community.

14.2. ParmEd

ParmEd (*parmed*) is a topology file editor written in Python that enables high level control of the primary force field file in Amber: the prmtop file. ParmEd will modify the topology file and produce a new topology file that will work with *sander*, *pmemd*, and NAB programs, and provides options unavailable otherwise. ParmEd currently supports topology files created with both *tleap* and *chamber* (but support is very limited for those created with *tinker_to_amber*).

NOTE: The ParmEd library was reorganized following the release of AmberTools 15, which necessitated changing the names of the scripts *parmed.py* and *xparmed.py* to *parmed* and *xparmed*, respectively (the .py suffix was dropped).

14.2.1. Running parmed

parmed is used in a manner very similarly to *cpptraj*.

```
usage: parmed [-h] [-v] [-i FILE] [-p <prmtop>] [-c <inpcrd>] [-O]
              [-l FILE] [--prompt PROMPT] [-n] [-e] [-s] [-r]
              [<prmtop>] [<script>]
positional arguments:
  <prmtop>              Topology file to analyze.
  <script>              File with a series of ParmEd commands to execute.
optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
Input Files:
  -i FILE, --input FILE
                        Script with ParmEd commands to execute. Default reads
                        from stdin. Can be specified multiple times to process
                        multiple input files.
  -p <prmtop>, --parm <prmtop>
                        List of topology files to load into ParmEd. Can be
                        specified multiple times to process multiple
                        topologies.
  -c <inpcrd>, --inpcrd <inpcrd>
                        List of inpcrd files to load into ParmEd. They are
                        paired with the topology files in the same order that
                        each set of files is specified on the command-line.
Output Files:
  -O, --overwrite       Allow ParmEd to overwrite existing files.
  -l FILE, --logfile FILE
                        Log file with every command executed during an
                        interactive ParmEd session. Default is parmed.log
Interpreter Options:
    These options affect how the ParmEd interpreter behaves in certain cases.
    --prompt PROMPT       String to use as a command prompt.
    -n, --no-splash       Prevent printing the greeting logo.
    -e, --enable-interpreter
                          Allow arbitrary single Python commands or blocks of
                          Python code to be run. By default Python commands will
                          not be run as a safeguard for your system. Make sure
                          you trust the source of the ParmEd command before
                          turning this option on.
  Error Handling:
    These options control how ParmEd handles various errors and warnings that
    appear occur during the course of Action execution
    -s, --strict          Prevent scripts from running past unrecognized input
                          and actions that end with an error. In interactive
                          mode, actions with unrecognized inputs and failed
                          actions prevent any changes from being made to the
                          topology, but does not quit the interpreter. This is
                          the default behavior.
    -r, --relaxed         Scripts ignore unrecognized input and simply skip over
                          failed actions, executing the rest of the script.
                          Unrecognized input in the interactive interpreter
                          emits a non-fatal warning.
```

Like with ptraj and cpptraj, if you do not supply the prmtop or the input_file, it will read the commands from STDIN as you type them.

14.2.2. ParmEd commands (they are all case-insensitive)

All actions that work on a topology file will use the “parm <idx>|<name>” input sequence to operate on a specified topology file. If present, either the topology file loaded <idx> topologies after the first one or the topology file loaded with the given <name> will be modified by that action. If absent, the LAST topology file loaded will be modified. The <idx> ranges from 0 to the total number of loaded topologies minus 1.

(Note: if you actually have a topology file named “1” that is not the second loaded topology file, you will need to address it via an index. That is, integers will always be assumed to be indices unless they are out of the topology file range.)

14.2.2.1. addAtomicNumber

Usage: *addAtomicNumber*

Adds a section in the topology file with the flag ATOMIC_NUMBER in order to identify specific elements. Elements are matched based on their atomic masses in the MASS section of the topology file. An atom is assigned an element by matching it with the element on the periodic table whose atomic mass is closest to the atom in question. This approach should work for any atom whose mass is either unchanged from the LEaP output or if that atom’s mass has only been changed to one of its isotopes.

14.2.2.2. addDihedral

Usage: *addDihedral <mask1> <mask2> <mask3> <mask4> <phi_k> <per> <phase> <scee> <scnb> [type <type>]*

Adds a dihedral term (will NOT replace an existing dihedral) between atoms in mask1, mask2, mask3, and mask4. The dihedral is defined around the bond between the atoms in mask2 and mask3. Each mask must define the same number of atoms. For mask1 defines atoms 1,2,3; mask2 defines atoms 11,12,13; mask3 defines atoms 21,22,23; and mask4 defines atoms 31,32,33, then 3 new dihedrals will be added. One between atoms 1, 11, 21, and 31, another between atoms 2, 12, 22, and 32, and a third between atoms 3, 13, 23, and 33. The dihedrals will be set with force constant *phi_k,* periodicity *per,* phase angle *phase*, 1-4 electrostatic scaling factor *scee* (this must be specified – the default Amber value is 1.2 and the default GLYCAM and CHARMM value is 1.0), the 1-4 van der Waals scaling factor *scnb* (this must be specified – the default Amber value is 2.0 and the default GLYCAM and CHARMM value is 1.0). The *type* is either "normal" or "improper". If this is an improper torsion, <*mask3*> should represent the central atoms bonded to all other atoms in the improper torsion.

End-group interactions are excluded automatically if the two end atoms (atoms 1 and 4) are bonded or angled to each other, or if they appear in a different dihedral. Otherwise, they are included. These are the same rules that *tleap* uses when it creates the topology file, and correctly accounts for complex exclusion rules involving ring systems (of size 4, 5, and 6) as well as multi-term torsion parameters.

14.2.2.3. addExclusions

Usage: *addExclusions <mask1> <mask2>*

Allows you to add arbitrary exclusions to the exclusion list. Every atom in <mask2> is added to the exclusion list for each atom in <mask1> so that non-bonded interactions between those atom pairs will not be computed. NOTE that this ONLY applies to direct-space (short-range) non-bonded potentials. For PME simulations, long-range electrostatics between these atom pairs are still computed (in different unit cells).

14.2.2.4. addLJType

Usage: *addLJType <mask> [radius <new_radius>] [epsilon <new_epsilon>] [radius_14 <new_radius14>] [epsilon_14 <new_epsilon14>]*

This command will assign all atoms specified in the given mask to a new van der Waals (VDW) atom type. Note that several different Amber atom types may in fact be the same VDW type, so this command is designed to give you control over changing just a single atom's (or single Amber atom type's) VDW parameters. Every atom specified in the mask will be given the SAME type (but different from every other atom in the topology file), even if their original VDW types are different. The parameters [new_radius] and [new_depth] are optional parameters that specify that atom's radius and well depth, which are combined with every other type's radius and depth via the canonical Amber combining rules. They default to the original value of the FIRST atom that is matched by the mask.

Note that for *chamber*-created topology files (ONLY), each atom type has separate 1-4 parameters that may be specified as well. Unspecified values will be taken from the default parameters of the first atom type as described above. Any attempt to supply the 1-4 parameters on a normal topology created with leap will result in an error.

See the command printLJTypes for additional information here. You can use this command to see if addLJType may be necessary for what you're trying to do.

14.2.2.5. addPDB

Usage: *addPDB <filename> [elem] [strict] [allicodes]*

This command replaces the *add_pdb* program that was released in previous AmberTools releases. It reads in a PDB file <*filename*> and adds the following new sections to the topology file:

RESIDUE_CHAINID The chain ID of each residue (if it was added by tleap and not in the PDB file, a * is used)

RESIDUE_ICODE PDB insertion code

RESIDUE_NUMBER The original residue number of this residue in the PDB file

ATOM_ELEMENT Atomic element. This section is redundant now that the topology file has an ATOMIC_NUMBER section. Therefore, this section is no longer printed by default.

The *strict* keyword turns residue mismatches (excluding solvent) into fatal errors. Note that for nucleic acids, terminal residue names often do not match the residue names in the PDB file because of the added 5 or 3 to the residue name (for the 5' terminus and 3' terminus, respectively).

The *elem* keyword will force the ATOM_ELEMENT section to be printed to the topology file, but the element will be determined from the ATOMIC_NUMBER section (or atomic mass if the former is not present) rather than the atom names as was done in the *add_pdb* program.

The *allicodes* keyword forces insertion codes to be printed even if every one will be blank. This allows parsers that use that section to be sure it will always be present.

Residues not in the PDB will be assigned a CHAINID of '*' and a RESIDUE_NUMBER of 0. While this action is based on, and reproduces the key results of, the historical *add_pdb* program, it is a bit more flexible.

14.2.2.6. add12_6_4

Usage: *add12_6_4 [<mask>] [c4file <c4file> | watermodel <watermodel>] [polfile <polfile>] [tunfactor <tunfactor>]*

The add12_6_4 command is designed to create the prmtop files, which contain the C_4 terms between the ions and each atom type in the prmtop file. By using it together with the outparm command, there will be a new flag named "LENNARD_JONES_CCOEF" created in the end of the output prmtop file. The C_4 terms between the ions and "OW" atom type (which is the oxygen atom of the water molecule, here we assume the polarizability of "HW" is equal to zero) has been determined. Detailed information can be found in the papers of Li, Merz and co-workers.[95–97]

The C_4 term between two different kinds of ions is calculated by following equation:

$$C_4(M-X) = tunfactor \times [\frac{C_4^M(H_2O)}{\alpha_0(H_2O)} \times \alpha_0(X) + \frac{C_4^X(H_2O)}{\alpha_0(H_2O)} \times \alpha_0(M)] \tag{14.15}$$

Where M and X mean two different kinds of ions which all have theirC_4 values towards "OW" determined. Herein $C_4^M(H_2O)$ and$C_4^X(H_2O)$ are the two C4 values.$\alpha_0(H_2O)$,$\alpha_0(M)$, and$\alpha_0(X)$ are the polarizabilities of "OW", "M", and "X", respectively.

The C_4 terms between every other atom pair including ion are calculated by following equation (including ion interact with itself, herein M represents ion, Y means the other atom type in the atom pair):

$$C_4(M-Y) = tunfactor \times \frac{C_4^M(H_2O)}{\alpha_0(H_2O)} \times \alpha_0(Y) \tag{14.16}$$

Here we explain the Usage terms:

1. The <mask> is the ion which was treated as the ion center when adding the C_4 terms. Please make sure its corresponding "ATOMIC_NUMBER" is correct in the prmtop file, in that this is the criterion used in the code to identify the metal ion. If you want to use 12-6-4 potentials for different kinds of ions in the prmtop file, please specify all of these ions in the <mask> together (i.e. <mask> contains at least one ion for each kind) other than performing several add12_6_4 commands for different kinds of ions one after another, because the the current implementation doesn't support the later situation. If no <mask> provided, default value :ZN.

2. To add the C_4 term between the ion and the "OW" atom type (the oxygen atom in the water molecules), you can either use your own <c4file> (in this way you need to create a <c4file> where the first column is the Atom Symbol plus charge and the second column is the corresponding C_4 value) or use the C_4 values stored in *parmed* (in this way you only need to specify the watermodel you are using in the command line, either TIP3P, SPCE or TIP4PEW). If nothing (neither your own <c4file> nor the water model you are using) is specified, the TIP3P water model will be treated as the default and the related values stored in *parmed* will be used.

3. To add C_4 terms between the ion and atom types besides "OW", you need a polarizability file of all the atom types (the default file is `$AMBERHOME/dat/leap/parm/lj_1264_pol.dat`, you can also create and use your own polfile where the first column is the Amber Atom Type while the second column is polarizability), and a tunfactor (which is shown in the previous equation - the default value is set as 1.0). The best tunfactor value for each force field may be different. You can also fine tune the optimal value for a specific force field if so inclined.

After using the *add12_6_4* command in the parmed, please don't forget to use the outparm command to output the new prmtop file. One thing need to clarify: for the <c4file>, the first column is Atom Symbol plus charge (e.g., Na1, Mg2, Cl-1, ...) and the second column is the C_4 value between the ion and the "OW" atom type; for the <polfile>, the first column is the AMBER Atom Type (e.g., HC, CT, N3, OS, Be2+, Mg2+, Ca2+ ...) while the second column is the polarizability.

14.2.2.7. cd

This changes into the given directory (just like the UNIX cd command).

14.2.2.8. chamber

Usage: chamber -top <RTF> -param <PAR> -str <STR> -psf <PSF> [-crd <CRD>] [-nocmap] [-box <a,b,c[,α,β,γ]>|bounding] [-radii <radiusset>]

CHAMBER (CHarmm↔AMBER) is a tool which enables the use of the CHARMM force field within AMBER's molecular dynamics engines (MDEs). If you make use of the CHARMM force field in Amber, please cite Ref. [67].

AMBER[25] and CHARMM[320, 321] are two approaches to the parametrization of classical force fields that find extensive use in the modeling of biological systems. The high similarity in the functional form of the two potential energy functions used by these force fields, Eq.(14.19 and 14.20), gives rise to the possible use of one force field within the other MDE.

$$\begin{aligned} V_{\text{AMBER}} &= \sum_{\text{bonds}} k\left(r-r_{eq}\right)^2 + \sum_{\text{angles}} k\left(\theta-\theta_{eq}\right)^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}[1+cos(n\phi-\gamma)]\ / \\ &+\sum_{i<j}\left[\frac{A_{ij}}{R_{ij}^{12}}-\frac{B_{ij}}{R_{ij}^{6}}\right]+\sum_{i<j}\left[\frac{q_iq_j}{\varepsilon R_{ij}}\right] \end{aligned} \tag{14.17}$$

$$\begin{aligned} V_{\text{CHARMM}} &= \sum_{\text{bonds}} k_b\left(b-b_0\right)^2 + \sum_{\text{angles}} k_\theta\left(\theta-\theta_0\right)^2 + \sum_{\text{dihedrals}} k_\phi\left[1+cos\left(n\phi-\delta\right)\right] \\ &+ \sum_{\text{Urey-Bradley}} k_u\left(u-u_0\right)^2 + \sum_{\text{impropers}} k\left(\omega-\omega_0\right)^2 + \sum_{\phi,\psi} V_{CMAP} \\ &+ \sum_{\text{nonbonded}} \varepsilon\left[\left(\frac{R_{\min_{ij}}}{r_{ij}}\right)^{12}-\left(\frac{R_{\min_{ij}}}{r_{ij}}\right)^{6}\right]+\frac{q_iq_j}{\varepsilon r_{ij}} \end{aligned} \tag{14.18}$$

For the implementation of the CHARMM force field within Amber, parameters that are of the same energy term can be directly translated. However, there are differences in the functional forms of the two potentials, with CHARMM having three additional bonded terms. With respect to the 1-4 non-bonded interactions, CHARMM scales these in a different manner: the electrostatic scaling factor (*scee*) is 1.0 in CHARMM but 1.2 in Amber, while the van der Waals scaling factor (*scnb*) is 1.0 within CHARMM but 2.0 in Amber. Additionally, CHARMM uses a different set of parameters in the Lennard-Jones equation for the van der Waals interaction if the two atoms are bonded 1-4 to each other.

The first additional bonded term is CHARMM's two-body Urey-Bradley term, which extends over all 1-3 bonds. The second is a four-body quadratic improper term. The final additional term is a cross term, named CMAP, [322, 323], which is a function of two sequential protein backbone dihedrals. This term originates from differences observed between classically calculated two-dimensional ϕ/ψ peptide free energy surfaces using the CHARMM22 force field and those of experiment. CMAP is a numerical energy correction which essentially transforms the 2D ϕ/ψ classical energy map to match that of a QM calculated map.

Support for these extra terms has required the development of extra sections to Amber's extensible prmtop format to accommodate this new information as well as modifications of the precision of existing sections. For example, the CHARMM parameter file stores the equilibrium angle (θ_0, Eq.14.20) parameter in degrees in its parameter file, while Amber stores it in radians in the prmtop. However, during the conversion with *chamber*, this becomes inexact when converted to radians. Within CHARMM this is done internally at runtime and the inexactness is determined by the variable type that will hold the result of this conversion. However, for Amber, this conversion is done at the *chamber* execution stage, and as a result is limited by the precision to which that specific parameter is written to the prmtop file. Hence the precision of the ANGLE_EQUIL_VALUE has been increased; similar changes were carried out for the CHARGE and VDW sections for the same reasons. Specifically, the modified sections of the prmtop format and the additions to it are as follows:

```
%FLAG CTITLE
```

The keyword CTITLE is used in place of TITLE to specify that this is a CHAMBER prmtop.

```
%FLAG FORCE_FIELD_TYPE
%FORMAT(i2,a78)
1 CHARMM 31 *>>>>>>>>CHARMM22 All-Hydrogen Topology File for Proteins <<
```

This section described the force field in use. The initial integer specifies the number of lines to be read. The keyword CHARMM here indicates that this is the CHARMM force field.

```
%FLAG CHARGE
%COMMENT Atomic charge multiplied by sqrt(332.0716D0) (CCELEC)
%FORMAT(3e24.16)
```

The default format for charge has been changed from 5e16.8 to 3e24.16

```
%FLAG CHARMM_UREY_BRADLEY_COUNT
%COMMENT V(ub) = K_ub(r_ik - R_ub)**2
%COMMENT Number of Urey Bradley terms and types
%FORMAT(2i8)
```

This additional section describes the number of CHARMM Urey-Bradley terms present and the total number of Urey-Bradley types in use.

```
%FLAG CHARMM_UREY_BRADLEY
%COMMENT List of the two atoms and its parameter index
%COMMENT in each UB term:  i,k,index
%FORMAT(10i8)
```

This additional section lists the atom indexes and parameter lookup index for each of the Urey-Bradley terms.

```
%FLAG CHARMM_UREY_BRADLEY_FORCE_CONSTANT
%COMMENT K_ub:  kcal/mole/A**2
%FORMAT(5e16.8)
```

This additional section lists the force constant for each of the Urey-Bradley types.

```
%FLAG CHARMM_UREY_BRADLEY_EQUIL_VALUE
%COMMENT r_ub:  A
%FORMAT(5e16.8)
```

This additional section lists the equilibrium value for each of the Urey-Bradley types.

```
%FLAG CHARMM_NUM_IMPROPERS
%COMMENT Number of terms contributing to the
%COMMENT quadratic four atom improper energy term:
%COMMENT V(improper) = K_psi(psi - psi_0)**2
%FORMAT(10i8)
```

This additional section lists the number of CHARMM improper terms present.

```
%FLAG CHARMM_IMPROPERS
%COMMENT List of the four atoms in each improper term
%COMMENT i,j,k,l,index i,j,k,l,index
%COMMENT where index is into the following two lists:
%COMMENT CHARMM_IMPROPER_{FORCE_CONSTANT,IMPROPER_PHASE}
```

```
%FORMAT(10i8)
```

This additional section lists the atom indices and index into the parameter arrays for each of the CHARMM improper terms.

```
%FLAG CHARMM_NUM_IMPR_TYPES
%COMMENT Number of unique parameters contributing to the
%COMMENT quadratic four atom improper energy term
%FORMAT(i8)
```

This additional section lists the number of types present for the CHARMM impropers.

```
%FLAG CHARMM_IMPROPER_FORCE_CONSTANT
%COMMENT K_psi:  kcal/mole/rad**2
%FORMAT(5e16.8)
```

This additional section lists the force constant for each CHARMM improper types.

```
%FLAG CHARMM_IMPROPER_PHASE
%COMMENT psi:  degrees
%FORMAT(5e16.8)
```

This additional section lists the equilibrium phase angle for each of the CHARMM improper types.

```
%FLAG LENNARD_JONES_ACOEF
%FORMAT(3e24.16)
```

The default format for the Lennard Jones A and B coefficients has been changed from 5e16.8 to 3e24.16.

```
%FLAG LENNARD_JONES_14_ACOEF
%FORMAT(3e24.16)
```

This additional section and the corresponding BCOEF section provide the alternative parameters for 1-4 VDW interactions in the CHARMM force field.

In concert with these prmtop additions, the appropriate modifications have to be made within *sander* and *pmemd* to enable the calculation of the energy and derivatives corresponding to these new terms. The intention behind the approach of creating a CHARMM enabled prmtop file is that the use of this prmtop file should be transparent to the user. Once a CHARMM prmtop file is produced by *chamber*, the *sander* and *pmemd* dynamics engines automatically detect the presence of CHARMM parameters in the prmtop file and automatically select the correct parameters and code paths.

WARNING: The use of an unpatched Amber molecular dynamics engine with a chamber-generated prmtop file will give undefined behavior, leading to incorrect results. If you see the following error at runtime:

```
ERROR: Flag "TITLE" not found in PARM file
```

it most likely means that you are using an old pmemd or sander executable.

CHAMBER (CHarmm↔AMBER) is a tool which enables the use of the CHARMM force field within AMBER's molecular dynamics engines (MDEs). If you make use of the CHARMM force field in Amber, please cite Ref. [67].

AMBER[25] and CHARMM[320, 321] are two approaches to the parametrization of classical force fields that find extensive use in the modeling of biological systems. The high similarity in the functional form of the two potential energy functions used by these force fields, Eq.(14.19 and 14.20), gives rise to the possible use of one force field within the other MDE.

$$\begin{aligned} V_{\text{AMBER}} &= \sum_{\text{bonds}} k\,(r-r_{eq})^2 + \sum_{\text{angles}} k\,(\theta-\theta_{eq})^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}[1+cos(n\phi-\gamma)]\ / \\ &\quad + \sum_{i<j}\left[\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}}\right] + \sum_{i<j}\left[\frac{q_iq_j}{\varepsilon R_{ij}}\right] \end{aligned} \tag{14.19}$$

$$\begin{aligned} V_{\text{CHARMM}} &= \sum_{\text{bonds}} k_b\,(b-b_0)^2 + \sum_{\text{angles}} k_\theta\,(\theta-\theta_0)^2 + \sum_{\text{dihedrals}} k_\phi\,[1+cos\,(n\phi-\delta)] \\ &\quad + \sum_{\text{Urey-Bradley}} k_u\,(u-u_0)^2 + \sum_{\text{impropers}} k\,(\omega-\omega_0)^2 + \sum_{\phi,\psi} V_{CMAP} \\ &\quad + \sum_{\text{nonbonded}} \varepsilon\left[\left(\frac{R_{\text{min}_{ij}}}{r_{ij}}\right)^{12} - \left(\frac{R_{\text{min}_{ij}}}{r_{ij}}\right)^{6}\right] + \frac{q_iq_j}{\varepsilon r_{ij}} \end{aligned} \tag{14.20}$$

For the implementation of the CHARMM force field within Amber, parameters that are of the same energy term can be directly translated. However, there are differences in the functional forms of the two potentials, with CHARMM having three additional bonded terms. With respect to the 1-4 non-bonded interactions, CHARMM scales these in a different manner: the electrostatic scaling factor (*scee*) is 1.0 in CHARMM but 1.2 in Amber, while the van der Waals scaling factor (*scnb*) is 1.0 within CHARMM but 2.0 in Amber. Additionally, CHARMM uses a different set of parameters in the Lennard-Jones equation for the van der Waals interaction if the two atoms are bonded 1-4 to each other.

The first additional bonded term is CHARMM's two-body Urey-Bradley term, which extends over all 1-3 bonds. The second is a four-body quadratic improper term. The final additional term is a cross term, named CMAP, [322, 323], which is a function of two sequential protein backbone dihedrals. This term originates from differences observed between classically calculated two-dimensional ϕ/ψ peptide free energy surfaces using the CHARMM22 force field and those of experiment. CMAP is a numerical energy correction which essentially transforms the 2D ϕ/ψ classical energy map to match that of a QM calculated map.

Support for these extra terms has required the development of extra sections to Amber's extensible prmtop format to accommodate this new information as well as modifications of the precision of existing sections. For example, the CHARMM parameter file stores the equilibrium angle (θ_0, Eq.14.20) parameter in degrees in its parameter file, while Amber stores it in radians in the prmtop. However, during the conversion with *chamber*, this becomes inexact when converted to radians. Within CHARMM this is done internally at runtime and the inexactness is determined by the variable type that will hold the result of this conversion. However, for Amber, this conversion is done at the *chamber* execution stage, and as a result is limited by the precision to which that specific parameter is written to the prmtop file. Hence the precision of the ANGLE_EQUIL_VALUE has been increased; similar changes were carried out for the CHARGE and VDW sections for the same reasons. Specifically, the modified sections of the prmtop format and the additions to it are as follows:

```
%FLAG CTITLE
```

The keyword CTITLE is used in place of TITLE to specify that this is a CHAMBER prmtop.

```
%FLAG FORCE_FIELD_TYPE
%FORMAT(i2,a78)
1 CHARMM 31 *>>>>>>>>CHARMM22 All-Hydrogen Topology File for Proteins <<
```

This section described the force field in use. The initial integer specifies the number of lines to be read. The keyword CHARMM here indicates that this is the CHARMM force field.

```
%FLAG CHARGE
%COMMENT Atomic charge multiplied by sqrt(332.0716D0) (CCELEC)
%FORMAT(3e24.16)
```

The default format for charge has been changed from 5e16.8 to 3e24.16

```
%FLAG CHARMM_UREY_BRADLEY_COUNT
%COMMENT V(ub) = K_ub(r_ik - R_ub)**2
%COMMENT Number of Urey Bradley terms and types
%FORMAT(2i8)
```

This additional section describes the number of CHARMM Urey-Bradley terms present and the total number of Urey-Bradley types in use.

```
%FLAG CHARMM_UREY_BRADLEY
%COMMENT List of the two atoms and its parameter index
%COMMENT in each UB term:  i,k,index
%FORMAT(10i8)
```

This additional section lists the atom indexes and parameter lookup index for each of the Urey-Bradley terms.

```
%FLAG CHARMM_UREY_BRADLEY_FORCE_CONSTANT
%COMMENT K_ub:  kcal/mole/A**2
%FORMAT(5e16.8)
```

This additional section lists the force constant for each of the Urey-Bradley types.

```
%FLAG CHARMM_UREY_BRADLEY_EQUIL_VALUE
%COMMENT r_ub:  A
%FORMAT(5e16.8)
```

This additional section lists the equilibrium value for each of the Urey-Bradley types.

```
%FLAG CHARMM_NUM_IMPROPERS
%COMMENT Number of terms contributing to the
%COMMENT quadratic four atom improper energy term:
%COMMENT V(improper) = K_psi(psi - psi_0)**2
%FORMAT(10i8)
```

This additional section lists the number of CHARMM improper terms present.

```
%FLAG CHARMM_IMPROPERS
%COMMENT List of the four atoms in each improper term
%COMMENT i,j,k,l,index i,j,k,l,index
%COMMENT where index is into the following two lists:
%COMMENT CHARMM_IMPROPER_{FORCE_CONSTANT,IMPROPER_PHASE}
%FORMAT(10i8)
```

This additional section lists the atom indices and index into the parameter arrays for each of the CHARMM improper terms.

```
%FLAG CHARMM_NUM_IMPR_TYPES
%COMMENT Number of unique parameters contributing to the
%COMMENT quadratic four atom improper energy term
%FORMAT(i8)
```

This additional section lists the number of types present for the CHARMM impropers.

```
%FLAG CHARMM_IMPROPER_FORCE_CONSTANT
%COMMENT K_psi:  kcal/mole/rad**2
%FORMAT(5e16.8)
```

This additional section lists the force constant for each CHARMM improper types.

```
%FLAG CHARMM_IMPROPER_PHASE
```

```
%COMMENT psi:  degrees
%FORMAT(5e16.8)
```

This additional section lists the equilibrium phase angle for each of the CHARMM improper types.

```
%FLAG LENNARD_JONES_ACOEF
%FORMAT(3e24.16)
```

The default format for the Lennard Jones A and B coefficients has been changed from 5e16.8 to 3e24.16.

```
%FLAG LENNARD_JONES_14_ACOEF
%FORMAT(3e24.16)
```

This additional section and the corresponding BCOEF section provide the alternative parameters for 1-4 VDW interactions in the CHARMM force field.

In concert with these prmtop additions, the appropriate modifications have to be made within *sander* and *pmemd* to enable the calculation of the energy and derivatives corresponding to these new terms. The intention behind the approach of creating a CHARMM enabled prmtop file is that the use of this prmtop file should be transparent to the user. Once a CHARMM prmtop file is produced by *chamber*, the *sander* and *pmemd* dynamics engines automatically detect the presence of CHARMM parameters in the prmtop file and automatically select the correct parameters and code paths.

WARNING: The use of an unpatched Amber molecular dynamics engine with a chamber-generated prmtop file will give undefined behavior, leading to incorrect results. If you see the following error at runtime:

```
ERROR: Flag "TITLE" not found in PARM file
```

it most likely means that you are using an old pmemd or sander executable.

This command will read topology information from a CHARMM or XPLOR PSF file and write an AMBER inpcrd and chamber-style prmtop file so the CHARMM force field can be run in *sander* and *pmemd*. PSF files generated by CHARMM, CHARMM-GUI, and VMD are all supported. PSF files are always generated using a set of topology (RTF) files that define the residues (akin to library files in tleap)—these files define the connectivity and atom types of all atoms in the system. Topology and parameter (PAR) files are always paired, so you must use the parameter file here that matches the topology file you used to create the PSF originally. This is *very* important. CHARMM stream files (that define both residue and parameter sections) are also supported, but must be specified using the -str flag. Do not pass any topology or parameter files to the -str flag. The -top and -param flags can be specified multiple times.

NBFIX terms defined in any stream or parameter files are read and implemented.

-top <RTF> CHARMM Residue Topology File (RTF). This is not needed if the atom types are defined in the parameter files (this seems to be true for CHARMM36 and probably later force fields).

-par <PAR> CHARMM Parameter file. This defines all of the CHARMM parameter files, and the one that corresponds to the topology file used to create the PSF must be used.

-str <STR> CHARMM stream file. Any parameters stored in the stream file will be loaded.

-toppar <RTF|PAR> CHARMM RTF, parameter, or stream file—the type is automatically detected from the name. All standard CHARMM files should work, but if you have changed the file name, you should use either the `-top`, `-par`, or `-str` flags above. Wild-cards are recognized, so you can do something like `-toppar toppar/*36_prot*` to get all of the files that contain `36_prot` in their name inside the `toppar` directory.

-psf <PSF> CHARMM/XPLOR/VMD Protein Structure File (PSF). This file defines the structure and topology of the system.

-crd <CRD> File containing coordinates for the system. CHARMM coordinate, restart, and PDB files are all supported. If this is a PDB file and a CRYST1 record defines a periodic box, the unit cell dimensions will be set from this information. You can use the -box argument to override this.

-nocmap Ignore any CMAP terms that may be defined. This is strongly discouraged unless you have a good reason to do it.

-box <a,b,c[$,\alpha,\beta,\gamma$**]>|bounding** Defines the periodic box dimensions (and will override any PBC defined in the coordinate file). You can either give the keyword "bounding," which will define the smallest possible orthorhombic box that encloses the centers of all atoms or you can give the unit cell dimensions. If you provide only lengths, the box shape is assumed to be orthorhombic.

-radii <radiusset> The AMBER implicit solvent radius set. The options are equivalent to the "set PBRadii <radiusset>" options in tleap. See page 228 for more information. Available choices are amber6, bondi, mbondi, mbondi2, and mbondi3. Default choice is "mbondi" (same as tleap).

Note, after using this command, the created parm object will be the active parm. You need to use either the parmout or outparm commands to actually print a topology file (and don't forget to also print a coordinate file!)

Validation Starting with version c36a2 of CHARMM, a command (**frcdump**) has been implemented which provides a validation route for alternate implementations of the CHARMM force fields. For a given system, this command writes the various force field potential energy contributions, as well as the energy gradient experienced by each atom, to a file using a specific format and to a high precision. The same formatted output can also be generated by the AMBER MDEs to facilitate comparison and to validate that the CHARMM force field is being implemented correctly in Amber's MDEs.

An example section of a charmm script that will write this output to a file called **charmm_gold_c36a2** is as follows:

```
open unit 20 form write name charmm_gold_c36a2
frcdump unit 20
close unit 20
```

The analogous mdin section for Amber is as follows:

```
&debugf
 do_charmm_dump_gold = 1,
/
```

Given this directive, the Amber MDE will stop after evaluating the potential energy of a system and write the energy and forces pertaining to this to a (hardcoded) file called **charmm_gold** in the same directory as the mdin file. The reader is invited to examine the various example test calculations within the *$AMBERHOME/test/chamber/dev_tests/* directory for in depth examples of the above. For such testing, it is recommended that both the CHARMM binary and the Amber MDE binaries be compiled with the same compiler. Given that CHARMM support within Amber and the *chamber* software is still somewhat experimental, the user is advised to carry out such a comparison before running a long production run.

Known limitations / Issues This is a non-exhaustive list of the current known bugs and/or limitations with *chamber*:

- CHARMM polarization models are not supported. (**IPOL /= 0**)
- The mdout file will contain extra potential energy fields pertaining to the CHARMM terms. This may break or confuse third party scripts that parse such outputs.
- Third party scripts and/or tools which do not correctly parse the extensible prmtop format may have issues with a *chamber*-generated prmtop file.

- The potential energy decomposition components (self, reciprocal, direct, adjusted) of the Particle Mesh Ewald energy generated in the **charmm_gold** file when the **do_charmm_dump_gold = 1** mdin option in Amber do not match with the breakdown used in CHARMM, however, the summation and resulting forces do match.

If other issues are found, the *parmed* authors would be very grateful if these could be reported to them, either via the Amber mailing list and/or directly to the authors. Please ensure that prior to reporting an issue, the *chamber* binary passes the test cases provided with AmberTools. Please provide a standalone example of the problem with all input files present and a script reproducing the sequence of commands that triggers the problem. The posting of large files (> 2 MB) to the Amber mailing list is not recommended; instead one should make the files available on a website somewhere and provide a link to it with the posting to the list.

14.2.2.9. change

Usage: *change <property> <atom_mask> <new_value>*

This command allows you to change the value of an atom's property for every atom in a given mask to a new value. The allowed atomic properties you can modify are the CHARGE (given in units of elementary atomic charges), MASS (in g/mol), RADII (in Angstroms, these are the GB radii), SCREEN (the GB screening parameters), ATOM_NAME, and AMBER_ATOM_TYPE (this is NOT the van der Waals type). Every atom in the mask will be given the same new_value.

NOTE: The prmtop utility used here stores the partial CHARGE array in terms of elementary atomic charges. All charges are multiplied by 18.2223 prior to being written to any new topology file (and is divided by that number when read in from a topology file). Therefore, if you are changing specific atomic charges in this case, specify new charges in elementary atomic charges.

NOTE: This command gives you access to specific atoms. If you want to change all of the GB radii to be compatible with a specific GB model, see the changeRadii command.

14.2.2.10. changeLJPair

Usage: *changeLJPair <mask1> <mask2> <Rmin> <epsilon>*

This command changes a specific pairwise interaction between the atom type of the atoms in mask1 (these must all be the same type) and the atoms in mask2 (these must all be the same type as well). Rmin and Depth are the pre-combined values of these variables, which allows you to define your own combining rules for a specific pair of atoms.

If you want to see which atoms this command will affect, you can use the printLJTypes with either of the given masks to get a list of atoms that share the same type as the atoms in that mask.

This command is similar to NBFIX available through CHARMM.

14.2.2.11. changeLJ14Pair

Usage: *changeLJ14Pair <mask1> <mask2> <Rmin> <epsilon>*

This command is similar to changeLJPair above, except it alters the 1-4 Lennard Jones terms only. Note that this command is only available for *chamber*-created topology files, and will result in an error if applied to a normal topology created with leap.

14.2.2.12. changeLJSingleType

Usage: *changeLJSingleType <mask> <Rmin> <epsilon>*

This command allows you to change the radius and well depth of particular nonbonded atom types. It will set new values for each interaction the selected type has with every other atom type (irrespective if changeLJPair altered one of these terms before).

14.2.2.13. changeProtState

Usage: *changeProtState <mask> <state #>*

Changes the protonation state of a residue that is titratable via constant pH simulations in Amber. <mask> must match all atoms of one, and only one, titratable residue as defined in $AMBERHOME/AmberTools/src/etc/cpin_data.py. As of Amber 11, current titratable residues include AS4, GL4, HIP, LYS, CYS, TYR, and the basic nucleic acid residues DAP, DCP, DG, DT, AP, CP, G, and U. See comments in cpin_data.py for descriptions of which state numbers correspond to which protonation charge state.

14.2.2.14. changeRadii

Usage: *changeRadii <parameter_set>*

Parameter set is one of the following: bondi, mbondi, mbondi2, mbondi3, amber6. This command will reset all of the intrinsic GB radii to the specified set without having to recreate a topology file through leap.

14.2.2.15. checkValidity

Usage: *checkValidity*

Thoroughly checks the topology file for a wide range of errors. It also checks for common mistakes, like missing disulfide bridges, for instance. More checks are done if a restart file is loaded prior to running this command. If you are getting a strange error from a simulation engine, it may be worth using this to check the prmtop. Note that this action, in particular, requires a version of Python 2.5 to 2.7.

14.2.2.16. defineSolvent

Usage: *defineSolvent <residue_list>*

This command will allow you to define custom solvent residues. The residue_list must be a comma-separated list with no whitespace separating the residue names. This is important for the proper determination of the SOLVENT_POINTERS and ATOMS_PER_MOLECULE sections of the topology file. By default, HOH and WAT residues are recognized as solvent.

14.2.2.17. deleteBond

Usage: *deleteBond <mask1> <mask2> [verbose]*

This command will delete all bonds in which one atom is in mask1 and the other atom is in mask2. It also deletes all other valence terms (angles, Urey-Bradleys,* torsions, impropers, and CMAPs,*) in which a deleted bond was a part. This is distinct from using setBond to assign a force constant of 0 because it also deletes other valence terms and removes those atoms from the respective nonbonded exclusion lists (since they are no longer bonded to each other).

If you use the “verbose” keyword, you will get a printout of every bond that is deleted.

*Some terms are only found in chamber-style topology files specifying a CHARMM force field.

14.2.2.18. deleteDihedral

Usage: *deleteDihedral <mask1> <mask2> <mask3> <mask4>*

Deletes the dihedral around <mask2> and <mask3> in which the end-groups are <mask1> and <mask4>. For multi-term dihedrals, it removes each term.

14.2.2.19. deletePDB

Usage: *deletePDB*

Deletes the flags that are added by *addPDB* (see description above).

14.2.2.20. energy

Usage: *energy [cutoff <cut>] [[igb <IGB>] [saltcon <conc>] | [Ewald]] [nodisper] [omm] [applayer] [platform <platform>] [precision <precision model>] [decompose]*

Computes the energy for a given structure. If you did not load a coordinate file on the command-line, you must use loadRestrt (see below) in order to load a set of coordinates (and box dimensions for periodic simulations). The options are:

cutoff <cut> The cutoff, in Angstroms, to use for the nonbonded cutoff. The default value is 1000 for non-periodic systems and 8 for periodic systems.

dumpfrc <filename> The file name to write atomic forces to. The format is a single header line starting with '#' followed by natom lines with the x, y, and z components of the force space-delimited.

Non-periodic options These options are applied only to non-periodic simulations. If the prmtop indicates periodicity (i.e., IFBOX > 0), these options are ignored.

igb <IGB> GB model to use. Allowed values are 0, 1, 2, 5, 6, 7, and 8. The values 0 and 6 indicate vacuum electrostatics. The other values match the options available in sander, pmemd, and NAB (see pages 59 and 821 for more details).

saltcon <conc> Salt concentration (in Molarity) to use when using GB implicit solvent. See page 61 for more information.

Periodic options These options are applied only to periodic simulations. If the prmtop does not indicate periodicity (i.e., IFBOX == 2), these options are ignored.

Ewald Use the Ewald sum to compute long-range electrostatics instead of Particle-Mesh Ewald (this is *much* slower than PME for large systems). This is equivalent to setting *ew_type=1* in *sander*.

nodisper Do not use the long-range dispersion correction to correct for Lennard-Jones interactions that are excluded beyond the cutoff. This is equivalent to setting *vdwmeth=0* in *sander* and *pmemd*.

OpenMM-specific options Instead of using the sander-Python API to compute energies and forces, you can use OpenMM. OpenMM must be installed and the Python application layer must be available for import. OpenMM cannot currently handle octahedral boxes (or any non-orthorhombic box).

omm This keyword must be present in order to use the OpenMM engine instead of the sander Python package. All following options are ignored unless this keyword is present

platform <platform> OpenMM compute platform to use. Options are CUDA, OpenCL, Reference, and CPU. Consult the OpenMM manual for more details.

precision <precision model> OpenMM precision model to use. Options are single, double, and mixed. Reference is always double and CPU is always single. The mixed precision model (default) uses single precision for calculations and double for accumulation.

decompose By default, OpenMM does not decompose energy contributions to different terms (e.g., bond, angle, torsion, etc.). If present, this keyword will make ParmEd break the energies down as much as possible (OpenMM does not compute non-bonded energy terms separately, so Lennard-Jones, 1-4 nonbonded interactions, and electrostatics will all be conflated into a single term). Energy terms are always decomposed when not using the OpenMM API.

applayer If present, this keyword will write a temporary topology file and load an OpenMM system using the support classes bundled with OpenMM directly (rather than using ParmEd's internal OpenMM System creator). This is provided as a way to validate the agreement between OpenMM's application layer and ParmEd.

14.2.2.21. go

Usage: *go*

Stop reading commands and execute every command that has come before. This has exactly the same effect as the End Of File (EOF) character. All commands in a script after “go” will be ignored. Placing “go” as the last line of a script is the same as not including it at all (since the next line contains EOF, which executes the same behavior). Thus, you can get the same behavior from the interactive session by either typing “go” or sending the EOF character (which on unix is CTRL-D)

14.2.2.22. gromber

Usage: gromber <top_file> [define <DEFINE[=VAR]>] [topdir <directory>] [radii <radiusset>]

Load a Gromacs topology file with parameters as an Amber-formatted system. Note, if your Gromacs topology file requires any include topology files (as most do), you will need to have Gromacs installed for this to work.

- `<top_file>`: The Gromacs topology file to load
- `<coord_file>`: The coordinate file to load into the system. Can be any recognized format (GRO, PDB, mmCIF, inpcrd, etc.)
- `define <DEFINE[=VAR]>`: Preprocessor defines that control the processing of the Gromacs topology file.
- `topdir <directory>`: The directory containing all Gromacs include topology files. This is only necessary if Gromacs is not installed in a location that ParmEd can find.
- `radii <radiusset>`: The GB radius set to use. Can be `mbondi`, `bondi`, `mbondi2`, or `amber6`. Default is `mbondi`

Gromacs topology files do not store the unit cell information. Therefore, in order to make sure that unit cell information is properly assigned to the resulting system, the provided `<coord_file>` should contain unit cell information (e.g., GRO, PDB, PDBx/mmCIF, and inpcrd files can all store box information).

ParmEd will try to locate the Gromacs topology directory using either the `GMXDATA` or `GMXBIN` environment variables (which should point to the `$PREFIX/share/gromacs` or `$PREFIX/bin` directories, respectively, where `$PREFIX` is the install prefix). If neither is set, the topology directory is located relative to the location of the *gmx* (Gromacs 5+) or *pdb2gmx* (Gromacs 4 or older) in the user’s `PATH`. If none of the above is true, the default installation location (`/usr/local/gromacs/share/gromacs/top`) is used. Any provided topdir will override default choices (but only for this particular command – future *gromber* actions will use the default location again).

You can provide as many defines as you wish, and the ordering you specify them is preserved. The default value assigned to each define is "1". To provide multiple defines, use the keyword multiple times, for example:

```
define MYVAR=something define MYVAR2=something_else ...
```

It is important to note that Gromacs supports a much larger array of bonded potentials than Amber does. Gromacs supports several different bonded potentials (cubic, quartic, and Morse, just to name a few), while Amber supports only the simple harmonic bond potential. Similarly, Amber only supports quadratic angle potentials, periodic torsions, and the 12-6 Lennard-Jones potential. Through the chamber extensions (see Sec. 14.2.2.8), Amber can also support Urey-Bradley angle potentials, quadratic improper potentials, and correction map (CMAP) potentials.

Other potential energy functions Gromacs supports—like the Buckingham nonbonded potential—cannot be computed in Amber and so will result in an error. ParmEd also currently only supports the Lorentz-Berthelot combining rules, although support for the geometric combining rules is planned.

14.2.2.23. HMassRepartition

Usage: *HMassRepartition [<mass>] [dowater]*

This action implements hydrogen mass repartitioning in which the mass of each hydrogen is changed to *<mass>* (the default value is 3.024 daltons if no mass is provided). The mass of the heavy atom that the hydrogen is attached to is adjusted so that the total mass remains the same. This allows longer time steps to be taken in dynamics (see the relevant literature regarding this approach; e.g., [324]). By default, partitioning is only applied to the solute since SHAKE on water is handled analytically (via the SETTLE algorithm). Water can be forcibly repartitioned using the keyword *dowater*.

14.2.2.24. help

Usage: *help [action]*

This command does one of two things. If *action* is not specified, a list of available commands along with their short usage statement is displayed in a nicely formatted table. If *action* is provided and that action exists, a usage statement along with a short description is printed. This is a useful reference for quick interactive sessions. You can use a single "?" character instead of the word 'help'.

14.2.2.25. history

Usage: *history*

This command prints a list of the previous commands that were run in ParmEd. This can be useful if you want to turn your interactive ParmEd session into a script (much like the *history* command works in the shell).

14.2.2.26. interpolate

Usage: *interpolate <nparm> [parm2 <other_parm>] [eleconly] [prefix <prefix>] [startnum<num>]*

This command can be useful to create topology files that are a linear combination of two topology files, specified by *<other_parm>* and the currently active parm (which can be set for this action using the *parm* keyword). If only two parms are loaded (see *listParms*, below), *<other_parm>* defaults to the inactive parm for this action.

The options are described below:

`<nparm>` Number of topology files that will be generated *in addition to* the two end-state parms.

`parm2 <other_parm>` The other topology file to use when interpolating prmtops (in addition to the active parm). The selection here works the same as the *parm* keyword for every other action.

eleconly If present, this only interpolates the charge vectors. This is currently the only supported mode, although van der Waals interpolation is planned for future versions.

prefix <prefix> The prefix of the prmtop file names that will be written by this action. Generated topologies will be written as `<prefix>.#`, where `#` starts from `<num>` (see below) and increases by 1 for each parm. Default is the name of the active parm.

startnum <num> The number to use as a suffix for the first generated parm. Default value is 1.

14.2.2.27. listParms

Usage: *listParms*

This command will list all of the topology file names for the topology files that have been loaded into the main list, highlighting the active one.

14.2.2.28. lmod

Usage: *lmod*

This action adjusts the Lennard Jones parameters to work with the LMOD code in Amber. It changes Lennard-Jones A-coefficients that are 0 to 1000 to improve numerical stability. This action replaces the *lmodprmtop* program.

14.2.2.29. loadCoordinates

Usage: *loadCoordinates <filename>*
This reads a coordinate file and loads the first ste of coordinates found into the active structure. File type is auto-detected, with supported file formats currently including:

- Amber restart file
- Amber NetCDF restart file
- CHARMM coordinate file
- CHARMM restart file
- Amber mdcrd trajectory file
- Amber NetCDF trajectory file
- PDB file
- PDBx/mmCIF file

For trajectories and PDB or mmCIF files with multiple models, the coordinates are taken from the first frame or model. Note, this is a generalization of the *loadRestrt* command, below.

14.2.2.30. loadRestrt

Usage: *loadRestrt <restart_filename>*
This command takes an inpcrd or a restart file to assign coordinates to each of the atoms. This is needed for any commands that require coordinates.

14.2.2.31. ls

This is supposed to emulate the Unix 'ls' program as closely as possible, and can be used inside ParmEd in the same way.

14.2.2.32. minimize

Usage: *minimize [cutoff <cut>] [[igb <IGB>] [saltcon <conc>]] [[restrain <mask>] [weight <k>]] [norun] [script <script_file.py>] [platform <platform>] [precision <precision model>] [tol <tolerance>] [maxcyc <cycles>]*
Uses OpenMM to minimize a structure. After this action, the coordinates stored in the topology file are updated with the minimized coordinates (and the minimized structure will be written if a coordinate file is provided in the *outparm* or *parmout* commands).

General options The following options apply to all systems

cutoff <cut> This is the non-bonded cutoff in Angstroms to use for the minimization. For periodic systems, the default value is 8 Angstroms. For non-periodic systems, no cutoff is applied.

restrain <mask> If provided, the given mask will have Cartesian positional restraints applied with the given force constant (see weight below)

weight <k> The restraint weight used in the positional restraints according to (14.21). Note that this force constant is not scaled by 1/2 as it is in Hooke's Law (so it is half the value of the corresponding force constant).

$$E_{restraint} = k\left(\mathbf{r} - \mathbf{r_{eq}}\right)^2 \tag{14.21}$$

norun Do not run the minimization—just write the script and quit. If no script is requested, an error is raised and nothing is done.

script <script_file.py> The name of a file in which to write a Python script that will perform the desired energy minimization using OpenMM.

tol <tolerance> The tolerance to use to determine when to stop the minimization. Default is 0.001.

maxcyc <cycles> The maximum number of minimization cycles to use. By default there is no limit—the minimization will run until the tolerance is reached.

Implicit Solvent Options The following options apply only to implicit solvent simulations

igb <IGB> GB model to use. Allowed values are 0, 1, 2, 5, 6, 7, and 8. The values 0 and 6 indicate vacuum electrostatics. The other values match the options available in sander, pmemd, and NAB (see pages 59 and 821 for more details).

saltcon <conc> Salt concentration (in Molarity) to use when using GB implicit solvent. See page 61 for more information.

OpenMM-specific options These options specify some computational details of the OpenMM calculation.

platform <platform> OpenMM compute platform to use. Options are CUDA, OpenCL, Reference, and CPU. Consult the OpenMM manual for more details. If you are using positional restraints, the CPU and Reference platforms will be even slower compared to the OpenCL and CUDA platforms than usual.

precision <precision model> OpenMM precision model to use. Options are single, double, and mixed. Reference is always double and CPU is always single. The mixed precision model (default) uses single precision for calculations and double for accumulation.

14.2.2.33. netCharge

Usage: *netCharge [mask]*

This command will calculate the net charge of all atoms belonging to a specific mask. If no mask is provided, it returns the net charge of all atoms in the topology file.

14.2.2.34. OpenMM

Usage: *OpenMM [sander/pmemd options] [-platform <platform>] [-precision <precision model>] [dcd] [progress] [script <script_file.py>] [norun]*

This action use OpenMM to run a molecular dynamics simulation in a mode very similarly to how sander or pmemd would run the same simulation. It recognizes all of the same command-line options as sander and pmemd in addition to the ones listed above. It will read an mdin file (given by the -i flag) and run an equivalent simulation (or as close to equivalent as possible) using the OpenMM Python application layer. See the OpenMM website (https://simtk.org/home/openmm) and manual for more details. If a simulation cannot be done, an error message is emitted.

The computational platform to use (CUDA, OpenCL, CPU, or Reference) can be provided as <*platform*>. By default, the fastest platform detected will be used. The precision model can be used to specify the precision of the variables that will be used as <*precision model*>. Currently supported options are “mixed” (single precision for calculations, double precision for accumulation), “single” (everything is done in pure single precision), and “double” (everything is done in pure double precision). As of OpenMM 6.0, only the CUDA and OpenCL platforms support multiple precision models. CPU is always single and Reference is always double.

The default prmtop that will be used is the active topology file, although either the *parm* or *-p* flags can be used to specify a different one. The *progress* keyword makes ParmEd print a message when it starts a new phase of the simulation.

The *script* keyword allows you to specify the name of a file in which a Python script that runs an equivalent calculation that ParmEd is running is printed to. This allows you to both inspect what ParmEd is doing behind-the-scenes with the OpenMM Python application layer as well as implement functionality not supported by Amber (but supported by OpenMM) without having to do the potentially laborious setup beforehand. The latter is particularly useful with the *norun* keyword, which will prevent ParmEd from running any dynamics or minimization.

The *dcd* keyword can be used to make ParmEd print the trajectory in DCD format. This is useful if you do not have a NetCDF Python package installed (any of scipy, ScientificPython, or netCDF4 will work), but still wish to generate a binary trajectory file.

See Chapter 17 and Chapter 18 for a more thorough description of the *sander/pmemd options*.

Some caveats for this action are listed below.

- The OpenMM package and the Python application layer must be installed and importable from the Python environment. ParmEd supports only OpenMM version 6.0 or higher.
- OpenMM itself requires Python 2.6 or later, which in turn passes on this requirement to this command in ParmEd.
- A NetCDF package for Python must be installed (for the Python interpreter used during the AmberTools configure step) and available to either read or write NetCDF trajectories and restarts. Supported NetCDF-Python packages are netCDF4, ScientificPython, or scipy (provided that the NetCDF bindings of those packages are included in the install). The scipy package is recommended.
- Trajectory file and restart file writing from the Python application layer are *very* slow, especially for ASCII versions of the files. NetCDF and DCD files are notably faster to write, but still incur significant overhead. Increasing the intervals for data printouts (ntpr, ntwx, ntwv, ntwf, and ntwr in the mdin file, for instance) can significantly improve computational performance, particularly for the GPU-enabled platforms.
- Not all features in sander and pmemd are supported, and not all unsupported options may be caught currently.

14.2.2.35. outCIF

Usage: *outCIF <file> [norenumber] [anisou]*

This will write a PDBx/mmCIF file from the currently active system. This is the new file format used by the Protein Data Bank in preference to the traditional PDB file format. The various options are described below:

<file> The name of the PDBx/mmCIF file to write

norenumber If this keyword is given, the original atom and residue numbering from the input structure are used rather than using the internal ordering used by Amber programs. If you used *addPDB* previously to add this information to the prmtop, this keyword will respect the numbering in the *original* PDB file. This will also work if you loaded your parm file from a PSF, PDB, or CIF file that may contain non-sequential numbering.

anisou If anisotropic B-factors are present, print them to the PDBx/mmCIF file.

14.2.2.36. outparm

Usage: *outparm <prmtop_name> [<restrt_name>]*

This command is just like parmout, except it can occur as many times as you want it to, and that topology file is written in the order in which that command is placed in the input file or read from STDIN (similar to outtraj in cpptraj). If you provide a file name for restrt_name, parmed will also write a valid restart file from the provided initial coordinates and velocities (if present) from the restart file added via the loadRestrt command. It will include velocities if they were present in the initial restart file. Note this is most useful when used in conjunction with the “strip” command. If all solvent is stripped, the box information will be discarded. If you do not strip all solvent molecules, the box info will remain unchanged from the original (even if you strip a large number of solvent molecules). If you removed a large number of solvent molecules, take care to re-equilibrate the density before continuing with production dynamics.

14.2.2.37. outPDB

Usage: *outPDB <file> [norenumber] [charmm] [anisou]*

This will write a PDBx/mmCIF file from the currently active system. This is the new file format used by the Protein Data Bank in preference to the traditional PDB file format. The various options are described below:

<file> The name of the PDBx/mmCIF file to write

norenumber If this keyword is given, the original atom and residue numbering from the input structure are used rather than using the internal ordering used by Amber programs. If you used *addPDB* previously to add this information to the prmtop, this keyword will respect the numbering in the *original* PDB file. This will also work if you loaded your parm file from a PSF, PDB, or CIF file that may contain non-sequential numbering.

charmm If a CHARMM SEGID identifier is loaded (either from the CHARMM PSF file or a CHARMM-modified PDB file), print that to the PDB file

anisou If anisotropic B-factors are present, print them to the PDB file as `ANISOU` records.

14.2.2.38. parm

Usage: *parm <filename>* | *parm set <filename>*|*<index>*

If used with the "set" keyword, the active topology is changed to the one with the given file name or the <index>+1'th topology file that was loaded. If used without the "set" keyword, it adds a new topology file to the list of available topologies from the given file name and sets that as the active topology for all future actions. (All previous actions were already applied to the previous 'active' topology).

14.2.2.39. parmout

Usage: *parmout <prmtop_name> [<restrt_name>]*

This command is similar to trajout in cpptraj and ptraj. It is ALWAYS the last command executed, and only the last parmout command is executed. It writes a topology file with all of the modifications made to it during the course of the whole ParmEd session. If you provide a file name for restrt_name, parmed will also write a valid restart file from the provided initial coordinates and velocities (if present) from the restart file added via the loadRestrt command. It will include velocities if they were present in the initial restart file. Note this is most useful when used in conjunction with the "strip" command. If all solvent is stripped, the box information will be discarded. If you do not strip all solvent molecules, the box info will remain unchanged from the original (even if you strip a large number of solvent molecules). If you removed a large number of solvent molecules, take care to re-equilibrate the density before continuing with production dynamics.

14.2.2.40. printAngles

Usage: *printAngles <mask> [<mask> [<mask>]]*

This will print out every angle that involves at least one atom specified by *<mask>*. If additional masks are given, only the angles in which the three atoms are specified in each of the given masks (with the central atom required to be in the second mask) are printed.

14.2.2.41. printBonds

Usage: *printBonds <mask> [<mask>]*

This will print out every bond that involves at least one atom specified by *<mask>*. If a second mask is given, only bonds in which one atom appears in each mask will be printed.

14.2.2.42. printDetails

Usage: *printDetails <mask>*

This command prints atomic details of every atom matching a given mask (atom number, residue number, residue name, atom name, atom type, van der Waals radius, van der Waals well depth, mass, and charge) in standard Amber units. This is a useful command to make sure that every atom you think belongs in a mask actually does belong in the mask (and that no atoms were missed). The mask parser implemented in Python here is (mostly) a copy of ptraj's mask parser implemented in C, but some parts had to be rewritten slightly to adjust for different syntaxes of the two languages. Note, distance-based criteria is not yet implemented in this parser.

14.2.2.43. printDihedrals

Usage: *printDihedrals <mask> [<mask> [<mask> [<mask>]]]*

This will print out every dihedral that involves at least one atom specified by <mask>. It labels dihedrals in which end-group interactions are omitted (either because they are in a multiterm dihedral or a ring) with an *M* and improper dihedrals with an *I* in the output. If multiple masks are given, only dihedrals that have one atom in each mask are printed. Ordering is important here, so the first atom must be in the first mask, the second atom in the second, etc. The order can be precisely reversed, but no other ordering is recognized.

14.2.2.44. printFlags

Usage: *printFlags*

This command prints every %FLAG present in the topology file (see http://ambermd.org/formats.html for a description of what each section labelled with these FLAGs means).

14.2.2.45. printInfo

Usage: *printInfo <flag>*

This command just prints out all of the data in a given prmtop %FLAG (see http://ambermd.org/formats.html for details)

14.2.2.46. printLJMatrix

Usage: *printLJMatrix <mask>*

This function prints out how every atom type interacts with the atom type(s) in <mask>.

14.2.2.47. printLJTypes

Usage: *printLJTypes [mask]*

This command prints out each atom's van der Waals, or Lennard-Jones type in the mask, as well as every other atom that shares the same atom type as any type in the mask. If no mask is provided, it prints out that information for every atom. This is particularly useful if you want to see if changing a particular pair interaction will affect more atoms than you expect. If it turns out that you wish to treat some of the atoms that share the same VDW type differently from one another, you will have to "separate" them by using the addLJType command before modifying them.

14.2.2.48. printPointers

Usage: *printPointers*

This command will print every pointer along with its name and a short description in the topology file. Solvated topology files will also have their SOLVENT_POINTERs printed in the same manner.

14.2.2.49. quit

Usage: *quit*

This command will halt *parmed* in its tracks. It is effectively the same as *go* except it will NOT execute any *parmout* command (although any *outparm* command used prior to quitting has already been executed)

14.2.2.50. scale

Usage: *scale <FLAG> <factor>*

This action scales all numbers in the *FLAG* section of the topology file by multiplying it by the number <*factor*>. This can be used, for instance, to scale all of the torsion force constants by a particular value in a Hamiltonian replica exchange simulation. [325]

14.2.2.51. scee

Usage: *scee <value>*

Allows the user to set/change the value of the electrostatic scaling constant that will be used to scale 1-4 electrostatic interactions. This needs to be set in the prmtop since it was removed from the *sander/pmemd* input file in Amber 11. This will apply <value> to all dihedral terms.

14.2.2.52. scnb

Usage: *scnb <value>*

Allows the user to set/change the value of the VDW scaling constant that will be used to scale 1-4 VDW interactions. This needs to be set in the prmtop since it was removed from the *sander/pmemd* input file in Amber 11. This will apply <*value*> to all dihedral terms.

14.2.2.53. setAngle

Usage: *setAngle <mask1> <mask2> <mask3> <k> <THETeq>*

Changes (or adds a non-existent) angle in the topology file. Each mask must select the same number of atoms, and an angle will be placed between the atoms in mask1, mask2, and mask3 (one angle between atom1 from mask1, atom1 from mask2, and atom1 from mask3, another angle between atom2 from mask1, atom2 from mask2, and atom2 from mask3, etc.)

14.2.2.54. setBond

Usage: *setBond <mask1> <mask2> <k> <Req>*

Changes (or adds a non-existent) bond in the topology file. Each mask must select the same number of atoms, and a bond will be placed between the atoms in mask1 and mask2 (one bond between atom1 from mask1 and atom1 from mask2 and another bond between atom2 from mask1 and atom2 from mask2, etc.)

14.2.2.55. setMolecules

Usage: *setMolecules [solute_ions=True|False]*

This command uses its own algorithm to determine system molecularity (which resets SOLVENT_POINTERS and ATOMS_PER_MOLECULE to what they *should* have been set to by *leap*). It will also determine if there are any errors in which molecules are not represented as consecutive atoms within a topology file (which won't happen unless you modify it yourself or there is a bug in tleap that prevents it from reordering atoms properly). However, in some unusual systems, tleap has been known to set the molecularity incorrectly, leading to strange segfaults and errors in sander and pmemd. Errors of this type can be caught with *checkValidity* and corrected using this command. It will also allow you to choose whether free ions are treated as part of the solute or part of the solvent.

14.2.2.56. setOverwrite

Usage: *setOverwrite [True|False]*

Allows the original topology file to be overwritten. By default, the original prmtop file is protected, and you cannot overwrite it. If you provide no value on this line, then it defaults to True. Note that no check is made if you are overwriting any other existing file (just the original topology).

14.2.2.57. source

Usage: *source <file>*

Loads a file with a list of ParmEd commands and executes them immediately.

14.2.2.58. strip

Usage: *strip <mask> [nobox]*

This will strip every atom that corresponds to the given atom mask out of the topology file altogether. Any bond, angle, or dihedral that it is a part of will be deleted as well. The bond, angle, and dihedral types that are no longer referenced after the atoms are stripped out are deleted from the topology file. All Lennard Jones parameters are kept, however, even if they are no longer used. In this way, any LJ modifications you did before the strip command will remain intact. The nobox keyword will make ParmEd delete the unit cell information from the topology file. This is necessary if you intend to use the resulting topology file for aperiodic simulations (e.g., using GB implicit solvent).

14.2.2.59. summary

Usage: summary

This command prints out a summary of topology file contents. If coordinates are present, more information is given (like system density). An example of the output is shown below:

Pure water:

```
Amino Acid Residues:   0
Nucleic Acid Residues: 0
Number of cations:     0
Number of anions:      0
Num. of solvent mols:  4096
Num. of unknown atoms: 0
Total charge (e-):     0.0000
Total mass (amu):      73793.5360
Number of atoms:       12288
Number of residues:    4096
System volume (ang^3): 122023.94
System density (g/mL): 1.004222
```

Implicit solvent protein system:

```
Amino Acid Residues:   108
Nucleic Acid Residues: 0
Number of cations:     0
Number of anions:      0
Num. of solvent mols:  0
Num. of unknown atoms: 0
Total charge (e-):     -4.0000
Total mass (amu):      11669.4360
Number of atoms:       1654
Number of residues:    108
```

14.2.2.60. tiMerge

Usage: *tiMerge <mol1mask> <mol2mask> <scmask1> <scmask2> [<scmask1N>] [<scmask2N>] [<tol>]*

This will remove redundant bonding terms and atoms from prmtop files for use in thermodynamic integration calculations with PMEMD. The input topology should have two molecules corresponding to V_0 and V_1. mol1mask/mol2mask are the atom masks for the molecules that should be merged (for V_0 and V_1 respectively). scmask1/scmask2 are the atom masks that list the unique atoms within the molecules to be merged. These do not necessarily have to be soft core atoms. For instance, removing the charges on a residue in a protein requires two copies of that residue in the prmtop file. These masks can be set to that residue. All atoms not in scmask1/scmask2 but in mol1mask/mol2mask should be the same, as these are considered common atoms. Any bonding terms which involve scmask atoms will be kept, but any extra terms will be removed. scmask1N/scmask2N are only used for atoms that will not be merged. These atoms will be included in the masks for output, so that additional soft core molecules that should not be merged do not have to be manually renumbered. tol specifies how close the coordinates have to be for the atoms in V_0 and V_1 to be considered the same. See Subsection 21.1.6 for a complete description of thermodynamic integration in PMEMD as well as an example of this command.

14.2.2.61. writeFrcmod

Usage: *writeFrcmod <frcmod_name>*

This command will dump a complete frcmod file containing every parameter in your topology file. (Note that the effects of a changeLJPair command will NOT be reflected in the topology file unless the pair you choose is between two atoms with the same VDW type, in which case it will alter *all* pair interactions with that atom type). It assumes the canonical Amber combining rules for VDW terms (Lorentz-Berthelot), and uses each type's interaction with itself to extract the well depths and VDW radii.

14.2.2.62. writeOFF

Usage: *writeOFF <OFF_File>*

Writes an Amber OFF (library) file containing every residue, including terminal residues, found in a given topology file. You must have loaded a coordinate file before running this command.

14.2.3. Examples

This section outlines a couple of example input files for *parmed* with comments describing what each command does. You can try these examples on the test parameter files in `$AMBERHOME/AmberTools/test/parmed` (either the `normal_prmtop/trx.prmtop` or the `chamber_prmtop/dhfr_gas.prmtop`).

Example 1

```
# This file generates a topology file with the new mbondi3 radii
# optimized for the igb = 8 GB model and changes the charge set
# of LYS 3 (trx.prmtop) to set up for a FEP-like calculation.
# In practice you would need more than just the protonated and
# deprotonated state (you would have to interpolate), but this
# is just a demonstration.

# Change to mbondi3
changeRadii mbondi3

# Output the first topology file
outparm trx_mbondi3_state0.parm7

# Change the charges of the LYS
change charge :3@N -0.3479
change charge :3@H 0.2747
change charge :3@CA -0.24
change charge :3@HA 0.1426
change charge :3@CB -0.10961
change charge :3@HB2,HB3 0.034
change charge :3@CG 0.06612
change charge :3@HG2,HG3 0.01041
change charge :3@CD -0.03768
change charge :3@HD2,HD3 0.01155
change charge :3@CE 0.32604
change charge :3@HE2,HE3 -0.03358
change charge :3@NZ -1.03581
change charge :3@HZ1 0
change charge :3@HZ2,HZ3 0.38604
change charge :3@C 0.7341
change charge :3@O -0.5894

# Output the second topology file
outparm trx_mbondi3_state1.parm7
```

Example 2

```
# This file generates a topology file in which the L-J
# interactions between atoms 10 and 28 have been removed,
# and the L-J interactions between atoms 40, 41, 42, and
# 57 with everybody else has been removed.

# Make atoms 10 and 28 new LJ types, but keep their original
# well depths and radii
addLJType @10
addLJType @28

# Zero the interaction between them
changeLJPair @10 @28 0.0 0.0

# Make atoms 40, 41, 42, and 57 a new LJ type with 0s for
# their parameters to remove all of their LJ interactions
# with every other atom
addLJType @40-42,57 radius 0.0 epsilon 0.0

# Write the final topology file. This statement could have
# been put anywhere
parmout altered_LJ.parm7
```

14.2.4. xparmed

To aid in simple tasks and make single- (or few-) prmtop file changes easier, a GUI version of ParmEd is available. It uses the Tk/Python graphical toolkit interface (called *Tkinter*). Tkinter is part of the standard Python library, but not all operating systems provide it with their system Python. The package names recognized by different package managers (e.g. *apt-get*, *port*, and *yum*) vary from system to system, and are detailed in the section below separated by common operating systems that have been tested by developers.

The GUI is very basic with a number of limitations. For instance, windows cannot be resized (but should fit on most standard terminals and should be sized appropriately). Furthermore, if an information window is present, the application will not end with the "*Exit xParmEd*" button until all information windows are closed. For scripting purposes, the text-based version, *parmed*, should be used instead.

14.2.4.1. Tkinter on Ubuntu (Debian)

To install Tkinter on Ubuntu (the package name on other Debians may differ), use the following command: *sudo apt-get install python-tk*

14.2.4.2. Tkinter on Red Hat

To install Tkinter on Red Hat (and CentOS and Fedora, probably), use the following command: *sudo yum install tkinter*

14.2.4.3. Tkinter on Mac OS X

The default Python installation on Mac OS X has Tkinter installed by default. In fact, it's a much 'prettier' version because it is built on top of Apple's GUI toolkits, which makes it look like a native Mac application. You can force Amber programs to use the Mac system Python by specifying /usr/bin/python as the default python to configure. If you wish to use a Python installed via MacPorts, you will need to also install the corresponding tkinter port. For instance, if you installed Python 2.7 from MacPorts and wish to use that, you will also need to install *py27-tkinter*.

14.2.4.4. Tkinter on Everything Else

If your system does not already have Tkinter installed, and none of the above helps you, you should consult a search engine or online forums. If it doesn't exist, you may have to stick with *parmed*.

14.2.5. Advanced Options

This section describes some of the advanced options in *parmed*. Note these are not generally available in *xparmed*

14.2.5.1. Interactive Python Shell

To increase ParmEd's flexibility, you can activate an limited, interactive Python interpreter to inject your own custom Python code into *parmed*'s normal execution. This brings with it the risk that custom code can be malicious if untrusted, so custom code evaluation is disallowed by default. To enable it, use the "-e" or "–enable-interpreter" command-line flag when executing *parmed*. To improve security, import statements are disallowed, although the math module has been imported for basic mathematical operations. To execute a single instruction, begin the command with a "!". In this case, leading whitespace is eliminated (so leading tabs/spaces are ignored here). For example,

```
bash $ parmed -e -n trx.prmtop
Loaded Amber topology file trx.prmtop

Reading input from STDIN...
> !print amber_prmtop.parm.parm_data['ATOM_NAME'][0:10]
['N', 'H1', 'H2', 'H3', 'CA', 'HA', 'CB', 'HB2', 'HB3', 'OG']
```

To execute a formatted block of code that requires more than one line, use "!!" to indicate to ParmEd that you wish to drop to interpreter mode. Terminate that block of code with another "!!" line. The prompt in STDIN-mode changes to "py >>>". For example:

```
bash$ parmed -e -n trx.prmtop
Loaded Amber topology file trx.prmtop

Reading input from STDIN...
> !!
```

```
py >>> def formatted_print(items):
py >>>    i = 0
py >>>    for item in items:
py >>>       print '%10.4f ' % item,
py >>>       i += 1
py >>>       if i % 5 == 0: print ''
py >>>    print ''
py >>>
py >>> formatted_print(amber_prmtop.parm.parm_data['CHARGE'][0:10])
py >>> !!
     0.1849      0.1898      0.1898      0.1898      0.0567
     0.0782      0.2596      0.0273      0.0273     -0.6714

> quit
Quitting.
```

The main topology class list being worked on is called amber_prmtop. The currently 'active' topology file is the 'parm' attribute of the list. You can also access specific topology files using an integer index or the original prmtop name. See the API documentation below if you are interested in making custom modifications. Note that it is VERY easy to break a topology file with this approach, so consider this an advanced option. A description of the topology file format can be found on http://ambermd.org/formats.html.

WARNING: Variable declarations you make here drop onto the top-level namespace in ParmEd's normal operating environment. That is, any variable you declare here MIGHT override a critical one for ParmEd. Variable names to avoid using include any of the Python built-in functions and types as well as *line, code, debug, actions, ParmError, LineToCmd, AmberParm, output_parm,* and *input*.

14.2.5.2. Extending ParmEd

This section describes what is necessary to add a new action to ParmEd.

All actions are parsed from the actions.py file in `$AMBERHOME/AmberTools/src/parmed/parmed/tools` directory. Each action must be its own class that inherits from `Action` and takes an `ArgumentList` as its first argument in its `init` method. All arguments should be extracted from the `ArgumentList` using its `get_next_<type>`, `get_key_<type>`, and `has_key` methods (the `get_key_<type>` and `has_key` methods should be called first). See existing methods as examples. You also need to take care to write the class doc-strings (the string immediately following every class declaration) to be as helpful as possible, because they are used in the help function. You must also add your command's usage statement in the `Usages` dictionary found at the top of `ParmedActions.py`, or it will be invisible to the help function and interpreter tab-autocompletion. The command name is taken as the first argument from that usage string.

No further action is necessary to add your functionality to ParmEd (and you should never have to edit `parmed` directly – any class put in `actions.py` is immediately accessible by `parmed` as long as it inherits from the `Action` base class). Existing actions provide helpful examples if you choose to expand ParmEd.

Extending xParmEd: Any action that is added to `actions.py` will be visible as buttons in `xparmed`, but will be disabled by default unless you implement that action directly. There is no well-defined standard for implementing actions in the GUI version like there is in the text-based version. GUI actions are defined in `$AMBERHOME/AmberTools/src/parmed/parmed/tools/gui/_guiactions.py`, and all additional actions must be defined there. You should only have to modify `_guiactions.py`, since the GUI is automatically sized and filled based on classes in `actions.py`. The best advice I can give if you want to expand xParmEd is to copy the class that does a similar task and modify it for your class. The related examples are fairly consistent in their style of implementation, so hopefully it is easy enough to add actions quickly.

14.2.5.3. ParmEd API

ParmEd is a rapidly changing program, and keeping comprehensive API documentation up-to-date is beyond the scope of this manual. Please see https://parmed.github.io/ParmEd for project documentation if you wish to use the ParmEd API in your own Python scripts. The documentation there is generated automatically from the source code and is kept up-to-date with the latest version. That said, you may find it useful to use some of the ParmEd commands described previously in your own Python scripts. This is described in the following paragraphs.

The actions in this version of ParmEd have been generalized to make it easy to incorporate them into your own Python scripts. To gain access to the actions, you must import them from the ParmedTools package. The Action class names are identical to the names printed in Subsection 14.2.2. When cast to a string, the action instance will output what it has done (or will do). The execute method bound to each Action instance will actually carry out the action on the specified topology file.

You can instantiate a new action in one of two ways, but the first argument must be an AmberParm (or ParmList) instance in both cases. Then, you can either load a single string with all of the options and key words (the same way as you would type it in *parmed*), or you can enter each argument independently with keywords being added appropriately.

An example showing how to add a new Lennard-Jones atom type is shown below using both techniques described above.

```
import os
import sys
from parmed.amber import AmberParm
from parmed.tools import addLJType

parm = AmberParm('trx.prmtop')

act = addLJType(parm, '@1 radius 0.0 epsilon 0.0')
act.execute()
print 'I just did:\n%s' % act

parm.writeParm('trx_modified.prmtop')

# The following code does the same thing
parm = AmberParm('trx.prmtop')

act = addLJType(parm, '@1', radius=0.0, epsilon=0.0)
act.execute()
print 'I just did:\n%s\n\t...again.' % act

parm.writeParm('trx_modified_2.prmtop')
```

15. Antechamber and GAFF

These are a set of tools to generate files for organic molecules and for some metal centers in proteins, which can then be read into LEaP. The Antechamber suite was written by Junmei Wang, and is designed to be used in conjunction with the general AMBER force field (GAFF) (gaff.dat).[326] See Ref. [327] for an explanation of the algorithms used to classify atom and bond types, to assign charges, and to estimate force field parameters that may be missing in gaff.dat. The Metal Center Parameter Builder (MCPB) program was developed by Martin Peters [328], and is described in Section 15.7. The python Metal Site Modeling Toolbox (pyMSMT) software package was developed by Pengfei Li, and is described in Section15.8.

Like the traditional AMBER force fields, GAFF uses a simple harmonic function form for bonds and angles. Unlike the traditional AMBER force fields, atom types in GAFF are more general and cover most of the organic chemical space. In total there are 33 basic atom types and 22 special atom types. The charge methods used in GAFF can be HF/6-31G* RESP or AM1-BCC.[329, 330] The force field parametrization was performed entirely with HF/6-31G* RESP charges. However, in most cases, AM1-BCC, which was parametrized to reproduce HF/6-31G* RESP charges, is recommended in large-scale calculations because of its efficiency. (Note that in AM1-BCC, the QM electrostatic potentials that were used as fitting targets were created in a very slightly different manner and then compared to RESP charges, using different scaling factors (i.e. 0.001/0.01 [330] versus 0.0005/0.001 [331].)

The van der Waals parameters are the same as those used by the traditional AMBER force fields. The equilibrium bond lengths and bond angles came from *ab initio* calculations at the MP2/6-31G* level and statistics derived from the Cambridge Structural Database. The force constants for bonds and angles were estimated using empirical models, and the parameters in these models were trained using the force field parameters in the traditional AMBER force fields. General torsional angle parameters were extensively applied in order to reduce the huge number of torsional angle parameters to be derived. The force constants and phase angles in the torsional angle parameters were optimized using our PARMSCAN package,[332] with an aim to reproduce the rotational profiles depicted by high-level *ab initio* calculations (geometry optimizations at the MP2/6-31G* level, followed by single point calculations at MP4/6-311G(d,p)).

By design, GAFF is a complete force field (so that missing parameters rarely occur); it covers almost all the organic chemical space that is made up of C, N, O, S, P, H, F, Cl, Br and I. Moreover, GAFF is totally compatible with the AMBER macromolecular force fields. It should be noted that GAFF atom types, except metal types, are in lower case, while AMBER atom types are always in upper case. This feature makes it possible to load both AMBER protein/nucleic acid force fields and GAFF without any conflict. One can even merge the two kinds of force fields into one file. The combined force fields are capable of studying complicated systems that include both proteins/nucleic acids and organic molecules. We believe that the combination of GAFF with AMBER macromolecular force fields will provide a useful molecular mechanical tool for rational drug design, especially in binding free energy calculations and molecular docking studies. Since its introduction, GAFF has been used for a wide range of applications, including ligand docking,[333] bilayer simulations,[334, 335] and the study of pure organic liquids [336].

15.1. Principal programs

The *antechamber* program itself is the main program of Antechamber. If your molecule falls into any of several fairly broad categories, *antechamber* should be able to process your PDB file directly, generating output files suitable for LEaP. Otherwise, you may provide an input file with connectivity information, i.e., in a format such as Mol2 or SDF. If there are missing parameters after *antechamber* is finished, you may want to run *parmchk2* to generate a frcmod template that will assist you in generating the needed parameters.

15.1.1. antechamber

This is the most important program in the package. It can perform many file conversions, and can also assign atomic charges and atom types. As required by the input, antechamber executes the following programs: *sqm* (or, alternatively, *mopac* or *divcon*), *atomtype*, *am1bcc*, *bondtype*, *espgen*, *respgen* and *prepgen*. It typically produces many intermediate files; these may be recognized by their names, in which all letters are upper-case. If you experience problems while running *antechamber*, you may want to run the individual programs that are described below (to facilitate this run antechamber with the option '-s 2').

Antechamber options:

```
-help print these instructions
-i input file name
-fi input file format
-o output file name
-fo output file format
-c charge method
-cf charge file name
-nc net molecular charge (int)
-a additional file name
-fa additional file format
-ao additional file operation
     crd : only read in coordinate
     crg: only read in charge
     radius: only read in radius
     name : only read in atom name
     type : only read in atom type
     bond : only read in bond type
-m multiplicity (2S+1), default is 1
-rn residue name, if not available in the input file
-rf residue topology file name in prep input file, default is molecule.res
-ch check file name in gaussian input file, default is molecule
-ek QM program (mopac or sqm) keyword (in quotes); overwrites previous keywords.
-gk gaussian keyword in a pair of quotation marks
-gm gaussian assign memory, inside a pair of quotes, such as "%mem=1000MB"
-gn gaussian assign number of processor, inside a pair of quotes, such as "%nproc=8"
-gv add keyword to generate gesp file (for Gaussian 09 only) 1: yes; 0: no, the default
-ge gaussian esp file generated by iop(6/50=1), default is g09.gesp
-df use divcon flag,  0 - use mopac; 2 - use sqm (the default)
-at atom type
    gaff : the default
    gaff2: for GAFF, version 2
    amber: for PARM94/99/99SB
    bcc  : for AM1-BCC
    sybyl: for atom types used in sybyl
-du check atom name duplications, can be yes(y) or no(n), default is yes
-bk 4-character component Id, for ccif
-j atom type and bond type prediction index, default is 4
     0 : no assignment
     1 : atom type
     2 : full bond types
     3 : part bond types
     4 : atom and full bond type
     5 : atom and part bond type
-eq equalize atomic charge, default is 1 for '-c resp' and '-c bcc'
     0 : no equalization
     1 : by atomic paths
     2 : by atomic paths and geometry, such as E/Z configurations
-s status information, can be 0 (brief), 1 (the default) and 2 (verbose)
-pf remove the intermediate files: can be yes (y) and no (n, default)
-pl maximum path length to determin equivalence of atomic charges for resp and bcc.
     The smaller the value, the faster the algorithm, default is -1 (use full length),
     set this parameter to 10 to 30 if your molecule is big (# atoms >= 100)
-dr acdoctor mode: validate the input file a la acdoctor, yes(y, default) or no(n)

-i -o -fi and -fo must appear in command lines and the others are optional

Use 'antechamber -L' to list the supported file formats and charge methods
```

List of the File Formats:

```
file format type  abbre. index | file format type abbre. index
--------------------------------------------------------------
Antechamber        ac       1  | Sybyl Mol2         mol2    2
PDB                pdb      3  | Modified PDB       mpdb    4
AMBER PREP (int)   prepi    5  | AMBER PREP (car)   prepc   6
Gaussian Z-Matrix  gzmat    7  | Gaussian Cartesian gcrt    8
Mopac Internal     mopint   9  | Mopac Cartesian    mopcrt 10
Gaussian Output    gout    11  | Mopac Output       mopout 12
Alchemy            alc     13  | CSD                csd    14
MDL                mdl     15  | Hyper              hin    16
AMBER Restart      rst     17  | Jaguar Cartesian   jcrt   18
Jaguar Z-Matrix    jzmat   19  | Jaguar Output      jout   20
Divcon Input       divcrt  21  | Divcon Output      divout 22
SQM Input          sqmcrt  23  | SQM Output         sqmout 24
Charmm             charmm  25  | Gaussian ESP       gesp   26
--------------------------------------------------------------
```

AMBER restart file can only be read in as additional file

List of the Charge Methods:

```
charge method     abbre.  index | charge method      abbre.
----------------------------------------------------------------
RESP               resp     1  |  AM1-BCC            bcc     2
CM1                cm1      3  |  CM2                cm2     4
ESP (Kollman)      esp      5  |  Mulliken           mul     6
Gasteiger          gas      7  |  Read in charge     rc      8
Write out charge   wc       9  |  Delete Charge      dc     10
----------------------------------------------------------------
```

Examples:

The basic use of *antechamber* is to pick input and output files and formats (via the `-i, -fi, -o, -fo` flags) , and choose various options for charge models, atom types, etc. A typical use would be:

```
     antechamber -i my.pdb -fi pdb -o my.mol2 -fo mol2 -c bcc -nc 1
```

The only “tricky” part is in generating resp charges, which requires interacting with the Gaussian program, and which varies depending on the version:

```
Using Gaussian 98 files as input:
(1) antechamber -i g98.out -fi gout -o sustiva_resp.mol2 -fo mol2 -c resp -eq 2
(2) antechamber -i g98.out -fi gout -o sustiva_cm2.mol2 -fo mol2 -c cm2

Using Gaussian03 files as input:
(11)antechamber -i g03.out -fi gout -o mtx.mol2 -fo mol2 -c resp
      -a mtx.pdb -fa pdb -ao name

Using Gaussian09 (version b1 and beyond):
(12)antechamber -i ch3I.mol2 -fi mol2 -o gcrt.com -fo gcrt -gv 1 -ge ch3I.gesp
    run Gaussian09 with gcrt.com as input
    antechamber -i ch3I.gesp -fi gesp -o ch3I_resp.mol2 -fo mol2 -c resp -eq 2
```

The following is the detailed explanations of some flags

-nc This flag specifies the net charge of the input molecule, otherwise, the net charge is read in from the input directly (such as gout, mopout, sqmout, sqmcrt, gcrt, etc.) or calculated by summing the partial charges (such as mol2, prepi, etc).

-a,-fa,-ao Sometimes, one wants to read additional information from another file other than the input, the '-ao' flag informs the program to read in which information from the additional file specified with '-a' flag. In Example (11), a mol2 file is generated from a Gaussian output file with atom names read in from a pdb file.

-ch,-gk,-gm,-gn Those flags specify the keywords and resource usage in Gaussian calculations

-ge,-gv The '-ge' flag specifies the file name of gesp file generated using iop(6/50=1) with Gaussian 09; the -gv flag specifies the Gaussian version and the default is '1' for Gaussian 09. If one wants to generate Gaussian input files (gcrt and gzmat) for older Gaussian versions, '-gv' must be set to '0'.

-rn The '-rn' line specifies the residue name to be used; thus, it must be one to three characters long.

-at This flag is used to specify whether atom types are to be created for the GAFF force field or for atom types consistent with parm94.dat and parm99.dat (i.e., the AMBER force fields). If you are using *antechamber* to create a modified residue for use with the standard AMBER parm94/parm99 force fields, you should set this flag to "amber"; if you are looking at a more arbitrary molecule, set it to "gaff", even if the molecule is intended for use as a ligand bound to a macromolecule described by the AMBER force fields.

-j This flag instructs the program how to run 'bondtype' and 'atom type'. '-j 1' assumes the bond types already exists; '-j 4' first predicts the connectivity table, then assigns bond and atom types sequentially; '-j 5' reads in connectivity table from the input and then run 'bondtype' and 'atomtype' sequentially. In most situations, '-j 4', the default option, is recommended. However, '-j 5' should be used if the input structure is not good enough and it includes the bond connectivity information (such as mol2, mdl, gzmat, etc.)

-eq This flag specifies how to do charge equilibration. With '-eq 1', atomic charge equilibration is predicted only by atom paths, in another word, if two or more atoms have exactly same sets of atom paths, they are equivalent and their charges are forced to be same. While '-eq 2' predicts charge equilibration using both atom paths and some geometrical information (E/Z configuration). With the '-eq 2' option, the charges of two hydrogen atoms bonded to the No 2 carbon of chloroethene are different as they adopt different configurations to chlorine (one is cis and the other is trans). Similarly, the two amide hydrogen atoms of acetamide do not share the same partial charge as the amide bond cannot rotate freely. To back-compatible to the older versions, the default is set to '1'

In Example (12), a gcrt file of iodine methane is generated and a gesp file named ch3I.gesp is produced when running Gaussian 09 with the default keyword. In Examples (13-15), RESP charges are generated for acetamide using different charge equilibration options. In the following table, the charges are listed for comparison purposes.

```
atom names        |      eq  =  0         |    eq  =  1        eq = 2
                  |no equalization        |atomic paths   | + geometry
--------------------------------------------------------------------------
methyl carbon     |-0.5190                | -0.5516        | -0.5193
methyl hydrogen   | 0.1412/0.1380/0.1396 |  0.1470        |  0.1397
carbonyl carbon   | 0.9673                |  0.9786        |  0.9673
oxygen            |-0.6468                | -0.6463        | -0.6468
nitrogen          |-1.1189                | -1.1219        | -1.1189
amide hydrogen    | 0.4556/0.4429         |  0.4501        | 0.4556/0.4429
--------------------------------------------------------------------------
```

15.1.2. parmchk2

parmchk2 reads in an ac/mol2/prepi/prepc file, an atomtype similarity index file (the default is $AMBERHOME/dat/antechamber/PARMCHK.DAT) as well as a force field file (the default is *$AMBERHOME/dat/leap/parm/gaff.dat*). It writes out a force field modification (frcmod) file containing any force field parameters that are needed for the molecule but not supplied by the force field (*.dat) file. Problematic parameters, if any, are indicated in the frcmod file with the note, "ATTN, need revision", and are typically given values of zero. This can cause fatal terminations of programs that later use a resulting prmtop file; for example, a zero value for the periodicity of the torsional barrier of a dihedral parameter will be fatal in many cases. For each atom type, an atom type corresponding file (ATCOR.DAT) lists its replaceable general atom types. By default, only the missing parameters are written to the frcmod file. When the "-a" switch is given the value "Y", *parmchk2* prints out all force field parameters used by the input molecule, whether they are already in the parm file or not. This file can be used to prepare the frcmod file used by thermodynamic integration calculations using sander.

Unlike *parmchk* which only checks several substitutions for a missing force field parameter, *parmchk2* enumerates all the possible substitutions and select the one with the best similarity score as the final substitue. Moreover, a penalty score, which measures the similarity between the missing force field parameter and the substitute is provided. The similarity socres are calculated using the similarity indexes defined in the atom type similarity index file (PARMCHK.DAT). A similarity index of a pair of atom types ('A/B') for a specific force field parameter type was generated by calculating the average percent absolute error of two set of force field parameters in gaff. The two set of force field parameters are identical except that one set has atom type 'A' and the other has 'B'. Each atom type pair ('A/B') has nine similarity indexes for nine different types of force field parameters, which are bond equilibrium length, bond stretching force constant, bond equilibrium angle ('A' and 'B' are central atoms), bond angle bending force constant ('A' and 'B' are central atoms), bond equilibrium angle ('A' and 'B' are non-central atoms), bond angle bending force constant ('A' and 'B' are non-central atoms), torsional angle twisting force constant ('A' and 'B' are inner side atoms), torsional angle twisting force constant ('A' and 'B' are outter side atoms), and improper dihedral angle.

```
parmchk2 -i  input file name
         -o  frcmod file name
         -f  input file format (prepi, prepc, ac, mol2)
         -p  ff parmfile
         -pf parmfile format,
             1: for amber FF data file (the default)
             2: for additional force field parameter file
         -c  atom type corresponding score file, default is PARMCHK.DAT
         -a  print out all force field parameters including those in the parmfile
             can be 'Y' (yes) or 'N' (no) default is 'N'
         -w  print out parameters that matching improper dihedral parameters
             that contain 'X' in the force field parameter file, can be 'Y' (yes)
             or 'N' (no), default is 'Y'
```

Example:

```
parmchk2 -i sustiva.prep -f prepi -o frcmod
```

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

15.2. A simple example for antechamber

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

As with any automated procedure, the output should be carefully examined, and users should be on the lookout for any unusual or incorrect program behavior.

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

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

(This file may be found at *$AMBERHOME/AmberTools/test/antechamber/tp/tp.pdb*). The basic command to create a mol2 file for LEaP is just:

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

The output file will look like this:

```
@<TRIPOS>MOLECULE
TP
   13    13     1     0     0
SMALL
bcc
@<TRIPOS>ATOM
      1 CG         -1.9590    0.1020    0.7950 ca     1 TP   -0.132000
      2 CD1        -1.2490    0.6020   -0.3030 ca     1 TP   -0.113000
      3 CD2        -2.0710    0.8650    1.9630 ca     1 TP    0.015900
      4 CE1        -0.6460    1.8630   -0.2340 ca     1 TP   -0.137000
      5 C6         -1.4720    2.1290    2.0310 ca     1 TP   -0.132000
      6 CZ         -0.7590    2.6270    0.9340 ca     1 TP   -0.113000
      7 HE2        -1.5580    2.7190    2.9310 ha     1 TP    0.136500
      8 S15        -2.7820    0.3650    3.0600 sh     1 TP   -0.254700
      9 H19        -3.5410    0.9790    3.2740 hs     1 TP    0.190800
     10 H29        -0.7870   -0.0430   -0.9380 ha     1 TP    0.133500
     11 H30         0.3730    2.0450   -0.7840 ha     1 TP    0.134000
     12 H31        -0.0920    3.5780    0.7810 ha     1 TP    0.133500
     13 H32        -2.3790   -0.9160    0.9010 ha     1 TP    0.136500
```

```
@<TRIPOS>BOND
     1    1    2 ar
     2    1    3 ar
     3    1   13 1
     4    2    4 ar
     5    2   10 1
     6    3    5 ar
     7    3    8 1
     8    4    6 ar
     9    4   11 1
    10    5    6 ar
    11    5    7 1
    12    6   12 1
    13    8    9 1
@<TRIPOS>SUBSTRUCTURE
     1 TP          1 TEMP              0 ****  ****    0 ROOT
```

This command says that the input format is pdb, output format is Sybyl mol2, and the BCC charge model is to be used. The output file is shown in the box titled .mol2. The format of this file is a common one understood by many programs. However, to display molecules properly in software packages other than LEaP and gleap, one needs to assign atom types using the '-at sybyl' flag rather than using the default gaff atom types.

You can now run parmchk2 to see if all of the needed force field parameters are available:

```
parmchk2 -i tp.mol2 -f mol2 -o frcmod
```

This yields the frcmod file:

```
remark goes here
MASS
BOND
ANGLE
DIHE
IMPROPER
ca-ca-ca-ha         1.1          180.0         2.0          General improper \\
                                        torsional angle (2 general atom types)
ca-ca-ca-sh         1.1          180.0         2.0          Using default value
NONBON
```

In this case, there were two missing dihedral parameters from the gaff.dat file, which were assigned a default value. (As gaff.dat continues to be developed, there should be fewer and fewer missing parameters to be estimated by parmchk2.) In rare cases, parmchk2 may be unable to make a good estimate; it will then insert a placeholder (with zeros everywhere) into the frcmod file, with the comment "ATTN: needs revision". After manually editing this to take care of the elements that "need revision", you are ready to read this residue into LEaP, either as a residue on its own, or as part of a larger system. The following LEaP input file (leap.in) will just create a system with thiophenol in it:

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

You can read this into LEaP as follows:

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

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

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

The procedure above only works as it stands for neutral molecules. If your molecule is charged, you need to set the -nc flag in the initial antechamber run. Also note that this procedure depends heavily upon the initial 3D structure: it must have all hydrogens present, and the charges computed are those for the conformation you provide, after minimization in the AM1 Hamiltonian. In fact, this means that you must have an reasonable all-atom initial model of your molecule (so that it can be minimized with the AM1 Hamiltonian), and you may need to specify what its net charge is, especially for those molecular formats that have no net charge information, and no partial charges or the partial charges in the input are not correct. The system should really be a closed-shell molecule, since all of the atom-typing rules assume this implicitly.

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

1. For the input molecules, make sure there are no open valences and the structures are reasonable. All hydrogen atoms must be present. Antechamber doesn't know what to do with metal ions (see the MCPB or MCPB.py program for that), or for other non-organic elements such as Boron. Look at the *$AMBERHOME/dat/leap/parm/gaff.dat* file to see what sorts of atomic environments are supported.

2. The Antechamber package produces two kinds of messages, error messages and informative messages. You may safely ignore those message starting with "Info". For example: "Info: Bond types are assigned for valence state 1 with penalty of 1".

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

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

5. *acdoctor* can diagnose many possible problems with input molecules. If you encounter failures when running antechamber programs, it is highly recommended to let *acdoctor* perform a diagnosis. Run the acdoctor program or use the acdoctor mode in program antechamber; the latter is controlled by option '-dr' and is on by default.

6. By default, the AM1 Mulliken charges that are required for the AM1-BCC procedure are computed using the *sqm* program, with the following keyword (which is placed inside the *&qmmm* namelist):

   ```
   qm_theory="AM1", grms_tol=0.0005, scfconv=1.d-10,
   ```

   For some molecules, especially if they have bad starting geometries, convergence to these tight criteria may not be obtained. If you have trouble, examine the *sqm.out* file, and try changing *scfconv* to 1.d-8 and/or increase the value of *grms_tol*. If you see failures in scf convergence that are not fixed by changing scfconv, try adding setting *ndiis_attempts*=700. You can use the `-ek` flag to antechamber to change these: for example

   ```
   antechamber ....  -ek "qm_theory='AM1', grms_tol=0.0005, scfconv=1.d-8, ndiis_attempts=700,"
   ....
   ```

   But be aware that there may be something "wrong" with your molecule if these problems arise; *acdoctor* may help (see the previous tip).

7. The standard procedure for obtaining AM1-BCC charges calls for a geometry optimization first. [329, 330] For some molecules (especially anions like phosphates) such a vacuum minimization may be inappropriate, since it can lead to formation of intramolecular hydrogen bonds that are not representative of the expected conformations in solution. If you trust your initial geometries, you can add *maxcyc*=0 to the `-ek` flag to skip the geometry minimization. You might also want to turn off geometry optimization in order try out several conformations in order to assess the sensitivity of the AM1-BCC charges to input geometry.

15.3. Using the components.cif file from the PDB

If your PDB file comes from the PDB itself (*www.rscb.org*), ligands and other molecules are called *components*, and you can download a `components.cif` file that contains a lot of useful information, including idealized geometries that include all hydrogen atoms. Antechamber can now read this sort of file (using the "-ccif" flag), so that you can create an initial forcefield like this:

```
antechamber -i components.cif -fi ccif -bk '02J' -o 02J.mol2 -fo mol2 -c bcc
```

In this example, we are preparing a mol2 file for residue "02J". Please note the following:

- The PDB has chosen to represent most components in the neutral form; in the above example, this implies that the carboxylic acid is protonated. If this is not what you want (probably not in this example), you will need to edit the mol2 file to remove the extra hydrogen, then re-run this back through antechamber. You may need to re-optimize the structure (say with *sqm*) before this second step. We are working to prepare a library with more useful protonation states; look for announcements on the Amber web site.
- You would also need to run *parmchk2* (as described above) to obtain any missing force field parameters.
- This is a new feature, and will certainly fail for some fraction of the 17,000 components that are in the `components.cif` file! Be sure to examine the outputs carefully, and expect to have failures.

15.4. Programs called by antechamber

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

15.4.1. atomtype

Atomtype reads in an ac file and assigns the atom types. You may find the default definition files in $AMBERHOME/dat/antechamber: ATOMTYPE_AMBER.DEF (AMBER), ATOMTYPE_GFF.DEF (general AMBER force field). ATOMTYPE_GFF.DEF is the default definition file. It is pointed out that the usage of atomtype is not limited to assign force field atom types, it can also be used to assign atom types in other applications, such as QSAR and QSPR studies. The users can define their own atom type definition files according to certain rules described in the above mentioned files.

```
atomtype -i input file name
         -o output file name (ac)
         -f input file format(ac (the default) or mol2)
         -p atom type set, supressed by "-d" option
            gaff  : the default
            amber : for PARM94/99/99SB
            bcc   : for AM1-BCC
            gas   : for Gasteiger charge
            sybyl : for atom types used in sybyl
         -d atom type definition file, optional
         -a do post atom type adjustment (it is applied with "-d" option)
            1: yes, 0: no (the default)
```

Example:

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

This command assigns atom types for sustiva_resp.ac with amber atom type definitions. The output file name is sustiva_resp_at.ac

15.4.2. am1bcc

Am1bcc first reads in an ac or mol2 file with or without assigned AM1-BCC atom types and bond types. Then the bcc parameter file (the default, BCCPARM.DAT is in $AMBERHOME/dat/antechamber) is read in. An ac file with AM1-BCC charges [329, 330] is written out. Be sure the charges in the input ac file are AM1-Mulliken charges.

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

Example:

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

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

15.4.3. bondtype

bondtype is a program to assign six bond types based upon the read in simple bond types from an ac or mol2 format with a flag of “-j part” or purely connectivity table using a flag of “-j full”. The six bond types as defined in AM1-BCC [329, 330] are single bond, double bond, triple bond, aromatic single, aromatic double bonds and delocalized bond. This program takes an ac file or mol2 file as input and write out an ac file with the predicted bond types. After the continually improved algorithm and code, the current version of bondtype can correctly assign bond types for most organic molecules (>99% overall and >95% for charged molecules) in our tests.

Starting with Amber 10, bond type assignment is proceeded based upon residues. The bonds that link two residues are assumed to be single bonded. This feature allows antechamber to handle residue-based molecules, even proteins are possible. It also provides a remedy for some molecules that would otherwise fail: it can be helpful to dissect the whole molecule into residues. Some molecules have more than one way to assign bond types; for example, there are two ways to alternate single and double bonds for benzene. The assignment adopted by bondtype is purely affected by the atom sequence order. To get assignments for other resonant structures, one may freeze some bond types in an *ac* or *mol2* input file (appending 'F' or 'f' to the corresponding bond types). Those frozen bond types are ignored in the bond type assignment procedure. If the input molecules contain some unusual elements, such as metals, the involved bonds are automatically frozen. This frozen bond feature enables bondtype to handle unusual molecules in a practical way without simply producing an error message.

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

Example:

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

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

15.4.4. prepgen

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

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

Examples:

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

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

15.4.5. espgen

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

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

Example:

```
(1) espgen -i sustiva_g98.out -o sustiva.esp
(2) espgen -i ch3I.gesp -o ch3I.esp
```

Command (1) reads in sustiva_g98.out and writes out sustiva.esp, which can be used by the resp program. Command (2) reads in a gesp file generated by Gaussian 09 and outputs the esp file. Note that this program replaces shell scripts formerly found on the AMBER web site that perform equivalent tasks.

15.4.6. respgen

Respgen generates the input files for two-stage resp fitting. Starting with Amber 10, the program supports a single molecule with one or multiple conformations RESP fittings. Atom equivalence is recognized automatically. Frozen charges and charge groups are read in with '-a' flag. If there are some frozen charges in the additional input data file, a RESP charge file, QIN is generated as well. Here are flags to *respgen*:

```
-i input file name(ac)
-o output file name
-l maximum path length (default is -1, i.e. the path can be any long)
-f output file format
   resp1 - first stage resp fitting
   resp2 - second stage resp fitting
   iresp1 - first stage i_resp fitting
   iresp2 - second stage i_resp fitting
   resp3 - one-stage resp fitting
   resp4 - calculating ESP from point charges
   resp5 - no-equalization
-e equalizing atomic charge (default is 1)
   0 not use
   1 by atomic paths
   2 by atomic paths and geometry (such as E/Z configuration)
-a additional input data (predefined charges, atom groups etc)
-n number of conformations (default is 1)
-w weight of charge constraint
   the default values are 0.0005 for resp1/iresp1 and 0.001 for
   resp2/iresp2
```

The following is a sample of additional respgen input file

```
//predefined charges in a format of (CHARGE partial_charge atom_ID atom_name)
CHARGE -0.417500 7 N1
CHARGE 0.271900 8 H4
CHARGE 0.597300 15 C5
CHARGE -0.567900 16 O2
//charge groups in a format of (GROUP num_atom net_charge),
//more than one group may be defined.
GROUP 10 0.00000
//atoms in the group in a format of (ATOM atom_ID atom_name)
ATOM 7 N1
ATOM 8 H4
ATOM 9 C3
ATOM 10 H5
ATOM 11 C4
ATOM 12 H6
ATOM 13 H7
ATOM 14 H8
ATOM 15 C5
ATOM 16 O2
```

Example:

```
respgen -i sustiva.ac -o sustiva.respin1 -f resp1
respgen -i sustiva.ac -o sustiva.respin2 -f resp2
resp -O -i sustiva.respin1 -o sustiva.respout1 -e sustiva.esp -t qout_stage1
resp -O -i sustiva.respin2 -o sustiva.respout2 -e sustiva.esp
        -q qout_stage1 -t qout_stage2
antechamber -i sustiva.ac -fi ac -o sustiva_resp.ac -fo ac -c rc -cf qout_stage2
respgen -i acetamide.ac -o acetamide.respin1 -f resp1 -e 2
respgen -i acetamide.ac -o acetamide.respin2 -f resp2 -e 2
```

The above commands first generate the input files (sustiva.respin1 and sustiva.respin2) for resp fitting, then do two-stage resp fitting and finally use antechamber to read in the resp charges and write out an ac file, *sustiva_resp.ac*. A more complicated example has been provided in *$AMBERHOME/test/antechamber/residuegen*. The last two 'respgen' commands generate resp input files for acetamide discriminating the two amide hydrogen atoms.

15.5. Miscellaneous programs

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

15.5.1. acdoctor

acdoctor reads the same input file formats used by the*antechamber* program and 'diagnoses' potential issues that can cause antechamber to fail. In AmberTools version 17 the acdoctor functionality was added to program antechamber; it is controlled by option '-dr' and is on by default. The first step is to validate some commonly-used molecular formats, such as pdb, mol2, mdl (sdf), etc. Then the presence of any unusual elements (elements other than C, O, N, S, P, H, F, Cl, Br and I) is reported. Unfilled valences are reported and additional checks are performed when atom types and/or bond types are read for file formats ac, mol2, sdf, prepi, prepc, mdl, alc and hin. The geometry is quantified by a distance matrix and atomic clashes are reported. *acdoctor* also applies a more stringent criterion than that utilized by *antechamber* to determine whether a bond is formed or not. A warning message is printed for those bonds that fail to meet the standard as well as for weird bonds. Next*acdoctor* determines whether all atoms are linked together through atomic paths. If not, an error message is printed. This kind of error typically implies that the input molecule has one or several bonds missing. Finally, *acdoctor* tries to assign bond types and atom types for the input molecule. If no error occurs during running *bondtype* and *atomtype*, presumably the input molecule should be free from problems when running the other Antechamber programs. It is recommended to diagnose your molecules with *acdoctor* when you encounter Antechamber program suite failures.

```
Usage: acdoctor -i  input file name
                -f  input file format
```

Example:

```
acdoctor -i test.mol2 -f mol2
```

The program reads test.mol2 and checks for potential problems when running the Antechamber programs. Errors and warning messages are printed. (Possible file formats are listed above in Section 15.1.1.

15.5.2. parmcal

parmcal is an interactive program to calculate the bond length and bond angle parameters, according to the rules outlined in Ref. [326].

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

15.5.3. residuegen

It can be painful to prepare a modified amino acid or nucleotide; the complication is that a residue is not a free standing molecule, and needs to be capped with extra atoms, usually at both termini. For “simple” systems, where a single conformation can be used to estimate partial charges, the *prepgen* program described above with the “-m” flag to specify which atoms to keep in the final residue. For more complex circumstances, the *residuegen* facilitates residue topology generation. *residuegen* reads in an input file and applies a set of antechamber programs to generate residue topologies in prepi format. The program can be applied to generate amino-acid-like topologies for amino acids, nucleic acids and other polymers as well. An example is provided below and the file format of the input file is also explained.

```
Usage: residuegen input_file
```

Example:

```
residuegen ala.input
```

This command reads in ala.input and generate residue topology for alanine. The file format of ala.input is explained below.

```
#INPUT_FILE:    structure file in ac format, generated from a Gaussian output
INPUT_FILE      ala.ac
#CONF_NUM:      Number of conformations utilized
CONF_NUM        2
#ESP_FILE:      esp file generated from gaussian output with 'espgen'
#               for multiple conformations, cat all CONF_NUM esp files onto ESP_FILE
ESP_FILE        ala.esp
#SEP_BOND:      bonds that separate residue and caps, input in a format of
#               (Atom_Name1 Atom_Name2), where Atom_Name1 belongs to residue and
#               Atom_Name2 belongs to a cap; must show up no more than two times
SEP_BOND        N1 C2
SEP_BOND        C5 N2
#NET_CHARGE:    net charge of the residue
NET_CHARGE      0
#ATOM_CHARGE:   predefined atom charge, input in a format of
#              (Atom_Name Partial_Charge); can show up multiple times.
ATOM_CHARGE     N1 -0.4175
ATOM_CHARGE     H4 0.2719
ATOM_CHARGE     C5 0.5973
ATOM_CHARGE     O2 -0.5679
#PREP_FILE:     prep file name
PREP_FILE:      ala.prep
#RESIDUE_FILE_NAME:     residue file name in PREP_FILE
RESIDUE_FILE_NAME:      ala.res
#RESIDUE_SYMBOL:        residue symbol in PREP_FILE
RESIDUE_SYMBOL:         ALA
```

15.5.4. match

The match program was developed to conduct least-square fittings for two molecules (one input and one reference) which are not necessarily the same in structure. Users can specify which atom or residue in the input corresponds to which in the reference in the definition file (-df). The users can also specify which atoms participating the fitting (-ds). The match matrix can be saved for translating and roating those atoms not participating the fitting procedure in separate step using '-j 2'.

```
Usage: match  -i  input file name
              -r  reference file name
              -f  format: 1-pdb (the default), 2-ac, 3-mol2, 4-sdf, 5-crd/rst
              -o  output file name
              -l  run log file name, default is "match.log"
              -s  selection mode
                  0: use all atoms (the default)
                  1: specify atom names
                  2: use atom definition file
                  3: use residue definition file - original residue IDs
                  4: use residue definition file - renumbered residue IDs
              -ds definition string if selection modes of '1' or '3' or '4'
                  e.g. 'C,N,O,CA', or 'HET' which stands for heavy atoms for '-ds 1')
              -df definition file if selection mode of '2' or '3' or '4'
                  records take a form of 'ATOM atom_id_input atom_id_reference'
                  or 'RES res_id_input res_id_reference'
              -n  number of atoms participating ls-fitting,
                  default is -1, which implies to use all the selected atoms
              -m  matrix file, default is "match.matrix"
              -t  job type:
                  0: calculate rms only, need -i and -r
                  1: lsfit, need -i, -r and -o the default
                  2: translation/rotation, need -i, -o and -m
```

Example:

```
match -f pdb -r 1be9.pdb -i 3pdz.pdb -o 3pdz_aligned.pdb -s 4 -ds "CA,C,N,O" -df 3pdz_1be9.corr
```

The program runs least-square fitting for the non-hydrogen main chain atoms of residues defined in the 3pdz_1be9.corr. A part of the 3pdz_1be9.corr is shown below:

RES 34 35 G G
RES 35 36 I I
RES 36 37 Y F
...
RES 87 88 L I
RES 88 89 L I

15.5.5. match_atomname

One limitation of the Antechamber package is that the atom name information is lost after running Gaussian calculations. And a residue topology file in prepi or prepc or a mol2 file generated from the Gaussian output has atom names not matching those from the original file (usually a pdb file). Because of this glitch, one can not simply load the residue topology file to tleap, read in the pdb file and then to save the topolgoy. We developed match_atomname to address this problem. The match_atomname program takes an input file and a reference file in pdb, ac, prepi, prepc and mol2 format, automatically detects the corresponding atom name in the reference for each atom name in the input. An output file in the same format as that of the input is generated using the matched atom names.

```
Usage: match_atomname -i  input file name
                      -fi input format (pdb, ac, prepi, prepc, mol2
                      -r  ref file name
                      -fr ref format (pdb, ac, prepi, prepc, mol2)
                      -o  output file name
                      -h  include hydrogen atoms or not
                          0 not, the default
                          1 yes
                      -g  geometric info (such as E/Z configuration) is considered to describe chem
                          0 no, the default
                          1 yes
                      -l  maximum path length, default is -1 (full length)
                          if it takes very long time and/or core dump occur, a value between 8 to 1
```

Example:

```
match_atomname -i SAH.prepi -fi prepi -o SAH_matched.prepi -r SAH_XRAY.pdb -fr pdb
```

The output, SAH_matched.prepi and SAH_XRAY.pdb can be loaded to tleap directly to generate a topology for minimization or MD simulations.

15.6. New Development of Antechamber And GAFF

One important of functions of Antechamber is to assign AM1-BCC charges for organic molecules. Openeye's Quacpak module can also assign AM1-BCC charges. The careful users may find that the charges assigned by the two programs are only marginally different (the largest charge difference is smaller than 0.05) in most cases. The difference is probably rooted from the difference of AM1 Mulliken charges. In unusual cases, large discrepancy occurs (the largest charge difference is larger than 0.1). Recently, we have systematically studied 585 marketed drugs using the both packages and the result is presented below. As the general AMBER force field is tightly related to the antechamber package, the new development of the GAFF is also summarized here.

15.6.1. Extensive Test of AM1-BCC Charges

Three methods, namely Antechamber/Mopac (Mulliken charges are calculated by Mopac), Antechamber/Sqm (Mulliken charges are calculated by sqm) and Openeye's Quacpak have been applied to assign the AM1-BCC charges for the 585 drug molecules. The first two methods give essentially similar charges for all the cases and the average charge difference is 0.005. The Quacpak, on the other hand, has an average charge difference of 0.015 to Antechamber/Mopac. When compared to RESP charges, the average charge differences are 0.102 and 0.105 for Antechamber/Mopac and Quacpak, respectively. In AM1-BCC, five BCC parameters were adjusted in order to improve agreement with the experimental free energies of solvation. Adjustments were made to bonds of amine nitrogen-H and amine nitrogen-tetravalent carbon.[329, 330] As a consequence, the average largest charge differences between AM1-BCC and RESP charges are very big: 0.441 for Antechamber/Mopac and 0.452 for Quacpak.

There are 71 molecules (12%) having the largest charge difference larger than 0.1 between Antechamber/Mopac and Quacpak. In comparison with the RESP charges, the average charge differences of the 71 molecules are 0.107 and 0.129 for Antechamber/Mopac and Quacpack, respectively. As to the average largest charge differences, the corresponding values are 0.444 and 0.522. It is clearly that Antechamber/Mopac-bcc has a similar average charge differences to RESP for the whole data set and the 71-molecule subset (0.102 vs 0.107), in contrast, Quacpac has a much larger average charge difference for the 71 molecules (0.129) than that of the whole data set (0.105). The similar trend is observed for the average largest charge difference as well (0.441 vs 0.444 for Antechamber/Mopac and 0.452 vs 0.522 for Quacpac).

15.6.2. New Development of GAFF

We have modified some parameters according to users' feedback. We would like to thank users who provide us nice feedback/suggestion, especially David Mobley and Gabriel Rocklin. This version (GAFF1.4) is a meta-version between gaff1.0 and gaff2.0 and the following is the major changes:

1. All the sp2 carbon in a AR2 ring (such as pyrrole, furan, pyrazole) are either 'cc' or 'cd' atom types (not 'c2' any more). This is suggested by Gabriel Rocklin from UCSF. This modification improves the planarity of multiple-ring systems

2. New van der Waals parameters have been developed for 'br' and 'i' atom types. The current parameters can well reproduce the experimental density data of CH_3Br (1.6755, 20 degree) and CH_3I (2.2789, 20 degree): 1.642 for CH_3Br and 2.25 for CH_3I, in contrast, the old parameters give 1.31 and 1.84, respectively.[336]

3. New van der Waals parameters have been suggested by David Mobley for 'c1', 'cg' and 'ch' atom types.[337]

4. We have performed B3LYP/6-31G* optimization for 15 thousands marketed or experimental drugs/bio-actives. Reliable bond length and bond angle equilibrium parameters were obtained by statistics: each bond length parameter must show up at least five times and has a rmsd smaller than 0.02 Å; each bond angle parameter must show up at least five times and has a rmsd smaller than 2.5 degrees. Those new parameters not showing up in old gaff were directly added into gaff 1.4; and some low-quality gaff parameters which show up less than five times or have large rmsd values (>0.02 Å for bond length and >5 degrees for bond angles) were replaced with those newly generated. In summary, 59 low quality bond stretching parameters were replaced and 56 new parameters were introduced; 437 low quality bond bending parameters were replaced and 618 new parameters were introduced.

15.7. Metal Center Parameter Builder (MCPB)

15.7.1. Introduction

The Metal Center Parameter Builder (MCPB) program provides a means to rapidly build, prototype, and validate MM models of metalloproteins. It uses the bonded plus electrostatics model to expand existing pairwise additive force fields. It was developed by Martin Peters at the University of Florida in the lab of Kenneth Merz Jr. MCPB is described fully Ref. [328].

Why is it desirable to model metalloprotein systems using MM models and more precisely within the bonded plus electrostatic model? Structure/function and dynamics questions that are not currently attainable using QM or QM/MM based methods due to unavailability of parameters or system size can be answered. Force fields have been developed for zinc, copper, nickel, iron and platinum containing systems using the bonded plus electrostatics model.

Incorporating metals into protein force fields can seem a daunting task due to the plethora of QM Hamiltonians, basis sets and charge models to choose from which the parameters are created. It was also generally carried out by hand without extensive validation for specific metalloproteins. MCPB was developed to remove the latter and create a framework in which to test various methods, basis sets and charges models in the creation of metalloprotein force fields.

The MCPB program was built using the MTK++ Application Program Interface (API). For more information regarding MTK++ and MCPB please see the MTK++ manual: *$AMBERHOME/AmberTools/src/mtkpp/doc/MTKpp.pdf* A more extensive description of metalloproteins and the theory within MCPB can be found in sections 10 to 12 of the MTK++ manual.

15.7.2. Running MCPB

MCPB takes two command-line arguments. One is the control file, which is required and chosen with the -i flag. The other is the log file, which is optional and chosen with the -l flag. A full listing of all the commands used by MCPB can be obtained with the -f flag.

```
MCPB: Semi-automated tool for metalloprotein parametrization
usage: MCPB [flags] [options]
options:
    -i script file
    -l log file
flags:
    -h help
    -f function list
```

Full details of a metalloprotein parametrization procedure using MCPB can be found in section 15.10 of the MTK++ manual. This example describes the active site parametrization of a di-zinc system (PDB ID: 1AMP). The parametrization is broken down into stages since several MCPB operations rely on the output of external packages such as Gaussian and RESP. Most of the steps are carried out using MCPB but some require user input and instruction.

15.8. Python Metal Site Modeling Toolbox (pyMSMT)

15.8.1. Introduction

The Python Metal Site Modeling Toolbox (pyMSMT) is a python package for metal site modeling of mixed systems (especially protein systems) for ultimate use in molecular dynamics simulations. It could facilitate parameterization of both the bonded and nonbonded models. This toolbox was developed by Pengfei Li in Prof. Kenneth M. Merz, Jr.'s research group at Michigan State University and now who is a postdoc in Prof. Sharon Hammes-Schiffer's research group at University of Illinois at Urbana-Champaign. Li and Merz have written a comprehensive review about metal ion modeling, which covers a wide spectrum of models including quantum mechanics models, classical force field models, polarizable force field models, reactive force fields, and some other types of models.[338] People who are interested can check the review article for more details. Users are welcome to send suggestions and bug reports to AMBER Mailing List (amber@ambermd.org).

In the current version, six applications are supported by the pyMSMT package:

1) MCPB.py: a Python version for the Metal Center Parameter Builder (MCPB). MCPB.py supports various metal ions (more than 80 metal ions with partial charges/oxidation-states from +1 to +8, literally), different AMBER force fields (ff94, ff99, ff99SB, ff03, ff03.r1, ff10, ff14ipq, ff14SB, ff14SB.redq, ff14SBonlysc, ff15ipq, ff15ipq-vac, fb15, GAFF, and GAFF2 in the current version), different parameterization methods (Seminario, Z-matrix and empirical), and different models (bonded model and nonbonded models for metal ions). It could facilitate parameterization of both metalloproteins and organometallic compounds. The workflow is more efficient and many of the modeling processes in previous MCPB versions are automatically implemented into MCPB.py (MCPB.py uses about 10 fewer steps and many fewer scripts than MCPB). An application note of the program is published by Li and Merz.[339] The main scheme and parameters are based on previous papers published by Merz et al.[94–97, 328]

2) IPMach.py: the Python ion parameterization machine. IPMach.py could largely facilitate the parameterization for the 12-6 LJ model and 12-6-4 LJ-type model of ions. It could automatically parameterize the 12-6 LJ model for a given hydration free energy or ion-oxygen distance, and the 12-6-4 LJ-type model for given hydration free energy and ion-oxygen distance. This code in AmberTools17 package now is in its beta-version.

3) PdbSearcher.py: the Python version of Pdbsearcher. PdbSearcher.py better supports the automatic recognition of the metal centers in a PDB file due to better compatibility with the PDB naming scheme of metal ions.

4) OptC4.py: a program to optimize theC_4 terms of the 12-6-4 potential using the AMBER topology and coordinate files. It can automatically optimize the metal-site-relatedC_4 terms to better reproduce the experimental structure. It uses the sum of unsigned error of metal site bond lengths, angles and dihedrals as the criterion (in which the bond, angle and dihedral have different weights). For each optimization cycle, the structure will be minimized by OpenMM[340, 341] and then have the sum of unsigned error calculated. It requires OpenMM version 6.3 and an installed SciPy package in current version.

5) CartHess2FC.py: the program to calculate the force constants using Cartesian Hessian matrix based on Seminario method. It could calculate all the bond and angle force constants of a system based on a Gaussian fchk file

or GAMESS log file which containing the Cartesian Hessian matrix.

6) espgen.py: the Python version of espgen in the antechamber package. It could extract the electrostatic potential information from a Gaussian output file or GAMESS log file which containing these information. It supports Gaussian03, Gaussian09 and GAMESS.

7) ProScrs.py: the "Protein Scissors" program for cutting and capping the protein segment into clusters.

8) car_to_files.py: the program to generate the mol2 and PDB files based on the car file. This function is designed for users of the INTERFACE force field in AMBER, which can be checked at https://bionanostructures.com/interface-md/.

15.8.2. Usage

The following is a summary of the usage and options for the three applications:

15.8.2.1. MCPB.py

```
Usage: MCPB.py -i input_file -s/--step step_number
               [--logf Gaussian/GAMESS-US output logfile]
               [--fchk Gaussian fchk file]
Options:
  -h, --help              Show this help message and exit
  -i INPUTFILE            Input file name
  -s STEP, --step=STEP    Step number
  --logf                  Gaussian/GAMESS-US output logfile
  --fchk                  Gaussian fchk file
```

The following is an introduction of the variables in the input_file:

(*Reminder:* there should be no blank lines in the input_file. The values or parameters should follow the variables separated by a space.)

Required variables:

ion_ids The PDB atom ID of the complex's central metal ion. It should be an integer value, depending on how many metal ions are included in the metal complex.

ion_info This variable is only required for the nonbonded model without refitting the residue charges (step number 4n2). In all, there are four data points required for each metal ion: 1) the residue name of the metal ion in the PDB file; 2) the atom name of the metal ion in the PDB file; 3) the element symbol of the metal ion; 4) the charge (or oxidation state, which needs to be an integer) of the metal ion. For example: ZN ZN Zn 2 (the first two are the residue and atom name of the Zn2+ ion in the PDB file, the third is its element symbol and the last one is its charge).

ion_mol2files The name(s) of the ion(s) in the mol2 file(s) contained in the metal center. This can be one or several name(s), depending on how many kinds of ions are included in the metal center. The user can use antechamber to transfer the single ion PDB file to a mol2 file and then manually modify the atom type and the atomic charge of the metal ion in the mol2 file.

original_pdb This is the file name of the original PDB file, which should have only one chain. The PDB file should have hydrogen atoms and metal ions in it. Users are advised to use an application like pdb4amber to clean up the PDB file first. They are also advised to add the hydrogen atoms by using a webserver such as H++ before performing the modeling in MCPB.py.

Optional variables:

add_bonded_pairs Specify the bonded atom pair(s) you want to add in the model building by MCPB.py. In default MCPB.py only detect the Metal-N/O/S/F/Cl/Br/I bond, if you have a other kind of metal ligating bond in the metal site (e.g. Metal-C bond), you need to specify the atom numbers of metal and the ligating atom in the input file. There should be even numbers after the variable with space seperate. For example, if you have two Metal-C bond in the metal site, and the metal has atomic number as 1001, while the two carbon atoms have atomic numbers as 1320 and 1380 respectively, you can use add following line in the input file: "add_bonded_pairs 1001 1320 1001 1380". [The default value of this variable is the null list.]

add_redcrd Specify whether additional redundant coordinates added to the Gaussian calculations for the small model. This option is designed for the Z-matrix method. If you are using Seminario method, this option can be ignored. In default Gaussian performed the geometry optimization using redundant internal coordinates, the default internal coordinates may not have the metal-ligand coordination bonds and angles (means the angles which including metal) included. If these bonds and angles are not included, it will cause users could not generate related force constants when using the Z-matrix method. 0 means do not add redundant coordinates for metal-ligand coordination bonds and angles. 1 means add redundant coordinates for metal-ligand coordinate bonds and angles to the optimization of the small model. In this way, the afterwards force constant calculation will use the same redundant internal coordinates as the optimization procedure when it reads the formerly generated chk file. Care should be taken that chosing 1 may cause convergence failure for geometry optimization, when chosing 2 is suggested. 2 means only do that for the force constant calculation of the small model. This option is suggested to use when user use option 1 but could not get a converged results for the geometry optimization procedure. In this way, Gaussian will perform the geometry optimization in a default manner, but the force constants for the Z-matrix will be based on the updated redundant coordinates. [The default is 0.]

additional_resids Specify the residues' IDs for which you want to add to the models built by MCPB.py. For example, it may be a residue in the second layer which coordinates a metal bonded residue. It will increase the computational cost for QM calculations. [The default value of this variable is the null list.]

anglefc_avg A variable used to indicate whether to make an average of angle force constants derived based on different manners of chosing the sub-matrices in Seminario method. There are A-B and B-A two ways of chosing the sub-matrix for two atoms in the parameter derivation process based on Seminario method. The angle force constant obtained based on different manners of chosing the sub-matrices may not have big differences. Two options are available: 0 or 1. 0 means not making average, using the default manner to chose the sub-matrices. 1 means making average of different manners to chose the sub-matrices. [The default is 0.]

bondfc_avg A variable used to indicate whether to make an average of bond force constants derived based on different manners of chosing the sub-matrix in Seminario method. There are A-B and B-A two ways of chosing the sub-matrix for two atoms in the parameter derivation process based on Seminario method. The bond force constant obtained based on different manners of chosing the sub-matrix would not have big differences. Two options are available: 0 or 1. 0 means not making average, using the default manner to chose the sub-matrix. 1 means making average of different manners to chose the sub-matrix. [The default is 0.]

chgfix_resids Specify the residues' IDs whose charges are going to be fixed during the charge fitting. The fixed charge values are referenced from the mol2 files used during the modeling. [The default value of this variable is the null list.]

cut_off The cutoff value is used to indicate there is a bond between the metal ion and the surrounding atoms. The unit is Angstroms. [The default is 2.8.]

force_field The user-designated name of the force field. The current version supports ff94, ff99, ff99SB, ff03, ff03.r1, ff10, ff14ipq, ff14SB, ff14SB.redq, ff14SBonlysc, ff15ipq, ff15ipq-vac, and fb15. [The default is ff14SB.]

frcmod_files The variable used to indicate the parameter modification file(s) for the nonstandard residue(s) (e.g. frcmod file generated by parmchk for a ligand molecule) in the metal complex. It can be one name or several names seperated by space. [The default value of this variable is the null list.]

gaff A variable used to indicate the use of a GAFF force field during the modeling. 0 means no, 1 means using GAFF, 2 means using GAFF2. [The default is 1.]

group_name The group name the user has specified. The group name is the prefix for different kinds of modeling files e.g. PDB, fingerprint and Gaussian input files for different models. [The default is MOL.]

ion_paraset The user-designated ion parameter set to be used in the nonbonded model. (This option has no influence on the metal ion VDW parameters in the bonded model, in which the author has chosen certain VDW parameter sets for different ions.) There are four options for this variable: HFE, CM, IOD and 12_6_4 (reminder: there are underlines between the numbers). If you use the 12-6 Lennard-Jones nonbonded model, the recommended settings are the HFE set for the +1 and -1 ions, the CM set for the +2 ions, and the IOD set for the +3 and +4 ions. They are also the default settings for these metal ions.

large_opt A variable used to indicate whether to do an geometry optimization in the Gaussian input file. Three options are available: 0, 1, or 2. 0 means no optimization, 1 means only optimizing the hydrogen positions, 2 means full geometry optimization. [The default is 0.]

lgmodel_chg Specify the total charge of the large model. [The default value of the charge will be assigned automatically by the program, which is not guaranteed to be right. Careful check is suggested from running the Gaussian/GAMESS-US program. If it is not right, you can add this option with right charge into the MCPB.py input file and regenerate the modeling files.]

lgmodel_spin Specify the spin of the large model. [The default value of the spin will be assigned automatically by the program as 1 or 2, based on the number of electrons. This is not guaranteed to be right. Careful check is suggested from running the Gaussian/GAMESS-US program. If it is not right, you can add this option with right spin into the MCPB.py input file and regenerate the modeling files.]

naa_mol2files The variable used to indicate non-amino acid mol2 file(s) in the metal complex if there are any nonstandard residue(s) in the metal complex. Examples of nonstandard residues include hydroxyl group and ligand molecules. For these residues, the user can use antechamber to generate the mol2 file(s) by first doing an AM1-BCC or HF/6-31G* RESP charge fit and then assigning an AMBER atom type (recommended for water or hydroxyl group) or a GAFF atom type (recommended for ligand). [The default value of this variable is the null list.]

scale_factor Specify the frequency scale factor for force constant derivation based on QM methods. This scale factor will scale all the bond and angle force constants determined from the QM calculations.*Reminder:* The force constant scale factor is usually equals the sqaure of frequency scale factor. For example, if you are using HF/6-31G* level of theory to do calculation and its frequency scale factor is 0.9, the force constant scale factor you need to use is 0.9^2=0.81. [The default value is 1.0 (no scaling performed).]

smmodel_chg Specify the total charge of the small model. [The default value of the charge will be assigned automatically by the program, which is not guaranteed to be right. Careful check is suggested from running the Gaussian/GAMESS-US program. If it is not right, you can add this option with right charge into the MCPB.py input file and regenerate the modeling files.]

smmodel_spin Specify the spin of the small model. [The default value of the spin will be assigned automatically by the program as 1 or 2, based on the number of electrons. This is not guaranteed to be right. Careful check is suggested from running the Gaussian/GAMESS-US program. If it is not right, you can add this option with right spin into the MCPB.py input file and regenerate the modeling files.]

software_version The version of software the user used to perform the QM calculations. Three options are available, g03 (which represents Gaussian03), g09 (which represents Gaussian09) and gms (which represents GAMESS-US). [The default is g03.]

sqm_opt A variable used to indicate the use of SQM in AmberTools to do a simulation of the sidechain and/or large model before using Gaussian to perform the calculation. *Please note:* if 1, 2 or 3 are chosen, the first step of the modeling process will take additional time (minutes for the sidechain model and hours for the large model). [The default is 0.]

- 0 – means no use of SQM.
- 1 – means the optimization is done only for the sidechain model.
- 2 – means the optimization is done only for the large model.
- 3 – means the optimization is done for both the sidechain and large models.

water_model The user-designated water model to be used in the molecular modeling. Options are TIP3P, SPCE and TIP4PEW. [The default is TIP3P.]

xstru Specify whether the structure in the original PDB file is used to generate the equilibrium bond distances and angle values in the frcmod file. 0 means not using, but use the QM optimized structure. 1 means using. [The default is 0.]

The following is an explanation of the step number variables:
Here are the options for the step_number:
For step1 there are three options: 1a (default, same as specifying 1), 1m and 1n.
For step2 there are four options: 2b, 2e, 2s (default, same as specifying 2) and 2z.
For step3 there are four options: 3a, 3b (default, same as specifying 3), 3c and 3d.
For step4 there are three options: 4b (default, same as specifying 4), 4n1 and 4n2.
The following is the detailed explanation of the steps used in the modeling procedure:

Step1. Used to generate the modeling files (e.g. PDB, fingerprint and Gaussian input files) for different models (e.g. sidechain, standard and large models). Three options are available and their explanation is shown below. Default is 1a.

- 1a – Used to automatically rename the atom types of the center metal ions and the surrounding bonded atoms in the standard fingerprint file.
- 1m – Used to automatically rename only the atom type(s) of the center metal ion(s) to the AMBER atomic ion atom type style in the standard fingerprint file.
- 1n – Used to generate the standard fingerprint file without renaming the atom types. Users can rename the atom type of the metal ion(s) and its ligating atoms manually in the standard fingerprint file.

Please note: Between using Step1 and Step2, the Gaussian calculations (if needed), should be done for the sidechain model (to calculate the force constants) and the large model (to do the RESP charge calculation) using Gaussian input files. Prior to the calculation, users can change the parameters (such as the calculation method, basis set, etc.) in the Gaussian input files according to their own preferences. After finishing this procedure, the user can move on to Step2.

Step2. Used to generate the frcmod file for the modeling. In this step, a frcmod file (with pre.frcmod name at the end of the file name), will be pre-generated. This file includes all the parameters, except the bond and angle parameters related to the metal ions. Later, the final frcmod file will be generated which will include all the parameters. There are three methods to choose from: Empirical, Seminario and Z-matrix. Each of these methods generates the metal ion-related bond and angle parameters. If you don't have a QM optimized structure, you can also generate a frcmod file with metal related bond and angle parameters as zero (as step 2b) and then manually modify it later for further usage. Default is the 2s (Seminario method).

- 2b - The "blank" methodGenerate a frcmod files with metal related bonds and angles have zero as the equilibrium values and force constants. If use with option "xstru 1" in the MCPB.py input file, it will generate the equilibrium values based on the orignal PDB structure and force constants as zero. User can modify the generated frcmod file by manually assigned bond and angle parameters for further usage.
- 2e – The Empirical method,[342]can generate the metal ion-related bond and angle parameters efficiently without doing Gaussian calculations. It only supports Zn2+ ion in the current version.

- 2s – The Seminario method[343] generates the force field parameters based on sub-matrices of the Cartesian Hessian matrix obtained from quantum calculations. This method requires a Gaussian fchk file (which can be generated from a chk file by using the formchk command in Gaussian).*Reminder:* both the geometry optimization and force constant calculation procedures are needed to generate the final chk file and subsequent fchk file for the force constant calculations done by the Seminario method.
- 2z – The Z-matrix method generates the force field parameters by using the Cartesian Hessian matrix obtained from the quantum calculations. This method requires the force constant Gaussian output file (usually named as a log file) after the geometry optimization and force constant calculations.

Step3. Used to perform the RESP charge fitting and to generate the mol2 files for the residues within the metal ion complex. There are several fitting schemes available in this step. The four options are shown below. The default is 3b since Seminario/ChgModB was identified as the best combination in the work of Peters et al.[328]*Reminder:* the chgfix_resids variable is effective in this procedure, if the variable is specified, the charge restriction will be used as well as one of the following choices.

- 3a – Allows all the charges of the atoms in the ligating residues to change without any restrictions.
- 3b – Restrains the charges of the heavy backbone atoms in the ligating residues according to the user-chosen force field.
- 3c – Restrains the charges of the backbone atoms (both heavy and hydrogen atoms) in the ligating residues according to the user-chosenforce field.
- 3d – Restrains the charges of the backbone atoms (both heavy and hydrogen atoms) and C beta atoms in the ligating residues according to the user-chosen force field.

Step4. Generates the leap input file. The default is 4b.

- 4b – Generates the leap input file for the bonded model.
- 4n1 – Generates the leap input file for the nonbonded model and refits the charge of the ligating residues.
- 4n2 – Generates the leap input file for the nonbonded model without refitting the charge of the ligating residues.

Here are some suggestions for the parameterization procedure:

1) For the modeling of the bonded model, the following steps are usually needed (4 steps): 1a/1n→2e/2s/2z→3a/3b/3c/3d→4b

2) For the modeling of a non-bonded model with a refitted charge, users can follow the workflow (3 steps): 1m→3a/3b/3c/3d→4n1

3) For modeling with a normal nonbonded model (without fitting any charges), users usually only need one step to perform the modeling (1 step): 4n2.

The following is an explanation of the logf and fchk variables:

These variables are optional. If provided, they are only active in step2 and/or step3. The default log file name is group_name + '_sidechain_fc.log' for step2 and group_name + '_large_mk.log' for step3. The default fchk file name is group_name + '_sidechain_opt.fchk' and it is only active for step2 when using Gaussian software and Seminario method to obtain the force constant parameters.

If you are using Gaussian software and Seminario method to generate force constant parameters, it uses the fchk file of sidechain model to store the Cartesian Hessian matrix. If you are using Gaussian software and Z-matrix method to generate force constant parameters, it uses the log file of sidechain model to store the force constant parameters. While if you are using GAMESS-US software and Seminario method to generate the force constants, it uses the log file of sidechain model to store the Cartesian Hessian matrix. In current version the software doesn't support GAMESS-US with Z-matrix method to generate the force constants.

Both the Gaussian and GAMESS-US software use the log file of large model to store the ESP charges.

15.8.2.2. IPMach.py

```
Usage: IPMach.py -i inputfile
Options:
  -h, --help      show this help message and exit
  -i INPUTF   Input file name
```

The following is an introduction of the variables in the input_file:

(*Reminder:* there should be no blank lines in the input_file. The values or parameters should follow the variables separated by a space.)

Required variables:

resname Residue name of the ion for parameterization (e.g. NA).

atname Atom name of the ion for parameterization (e.g. NA).

element Element of the ion for parameterization (e.g. Na).

attype Atom type of the ion for parameterization (e.g. Na+).

attype Charge of the ion for parameterization (e.g. 1).

Optional variables:

cpus Used cpu numbers during the parameterization (e.g. 2). There are at least 2 cpus needed to perform TI simulation using sander program. While 1 cpu could also be used to perform TI simulation using pmemd program. [The default is 2.]

gpus Used gpu numbers during the parameterization (e.g. 0). The gpus could only be used in MD simulation to obtain the ion-oxygen distance and coordination number values of the first solvation shell but could not used to perform the TI simulation to obtain the hydration free energy. This is due to the current version of pmemd.cuda doesn't support TI simulation yet. [The default is 0.]

tisteps Number of steps used during the thermodynamic integration simulation. Two options available: 1 or 2. 1 means using the "one-step" method while 2 means using the "two-step" method. The later one would need longer simulation time but give better accuray. [The default is 2.]

ti_windows Number of windows used for the "one-step" TI simulation. Which is not effective when using the "two-step" TI simulation method. [The default is 7.]

vdw_windows Number of windows used for the VDW scaling step in the "two-step" simulation method. Which is not effective when using the "one-step" TI simulation method. [The default is 3.]

chg_windows Number of windows used for the charge scaling step in the "two-step" simulation method. Which is not effective when using the "one-step" TI simulation method. [The default is 7.]

rev Whether reverse TI simulation performed. There are two options available: 0 or 1. 0 means no, 1 means yes. Performing reverse TI simulation could double the simulation time but offer more valid results. [The default is 1.]

rmin VDW radius parameter (unit is Angstrom) used for the initial guessing. [The default is 1.5.]

hfe Target hydration free energy value (unit is kcal/mol) used for the parameterization. [The default is -100.0.]

hfe_tol Tolerence of target hydration free energy value (unit is kcal/mol) during the parameterization. [The default is 1.0]

iod Target ion-oxygen distance of first solvation shell (unit is Angstrom) used for the parameterization. [The default is 2.0.]

iod_tol Tolerence of target ion-oxygen distance of first solvation shell (unit is Angstrom) during the parameterization. [The default is 0.01.]

cal_type Type of the calculation: whether it is a optimization (OPT), or single point calculation (SP). [The default is OPT.]

set Parameter set to generate. There are two parameter set available: HFE or IOD. The HFE parameter set will treat hydration free energy as target during the parameterization. The IOD parameter set will treat ion-oxygen distance of first solvation shell as target during the parameterization. [The default is HFE.]

maxiter Maximum iteration steps during the parameter optimization. [The default is 100.]

mode There are three modes available for running the program: test, scan, normal. These three modes will increase the simulation time one-by-one. [The default is normal]

program The MD program used during the parameterization. Two options: sander or pmemd. Reminder: pmemd could not used to perform TI simulation for 12-6-4 LJ-type model. [The default is sander.]

watermodel Water model used during the parameterization. The ion parameters may vary for different water models. It is suggested to parameterize a ion model for a specific water model. Ten options: tip3p, tip4p, tip4pew, tip5p, spc, spce, opc3, opc, fb3, and fb4. [The default is tip3p.]

c4 Initial C4 values between ion and oxygen in water for parameterization of the 12-6-4 model for ions. Its unit is kcal/mol * Angstrom^4. [The default is 0, means only 12-6 parameters with no C4 parameters applied.]

15.8.2.3. PdbSearcher.py

```
Usage: PdbSearcher.py -i/--ion ionname -l/--list input_file
                      -e/--env environment_file -s/--sum summary_file
                      [-c/--cut cutoff]
Options:
  -h, --help                 Show this help message and exit
  -i IONNAME, --ion=IONNAME  Element symbol of ion, e.g. Zn
  -l INPUTF, --list=INPUTF   List file name, list file contains one
                             PDB file name per line
  -e ENVRMTF, --env=ENVRMTF  Environment file name. An environment file is used
                             to store the metal center environment information
                             such as ligating atoms, distance, geometry etc.
                             For each bond, there is a record.
  -s SUMF, --sum=SUMF        Summary file name. A summary file is used to store
                             the metal center summary information such as metal
                             center geometry, ligating residues etc. For each
                             metal center there is a record.
  -c CUTOFF, --cut=CUTOFF    Optional. The cut off value used to detect the
                             bond between metal ion and ligating atoms.
                             The unit is Angstroms. If there is no value
                             specified, the default algorithm will be used.
                             The default algorithm recognizes the bond when
                             its distance is no less than 0.1 (smaller than 0.1
                             usually indicates a low quality structure) and no
                             bigger than the covalent radius sum of the two
                             atoms with a tolerance of 0.4.
```

15.8.2.4. OptC4.py

```
Usage: OptC4.py -m amber_mask -p topology_file  -c coordinate_file -r restart_file
                [--maxsteps maxsteps] [--phase simulation_phase]
                [--size optimization_step_size] [--method optimization_method]
                [--platform device_platform] [--model metal_complex_model]
Options:
 -h, --help                  Show this help message and exit
 -m ION_MASK                 Amber mask of the center metal ion
 -p PFILE                    Topology file
 -c CFILE                    Coordinate file
 -r RFILE                    Restart file
 --maxsteps=MAXSTEPS         Maximum minimization steps performed by OpenMM
                             in each parameter optimization cycle.
                             [Default: 1000]
 --phase=SIMUPHA             Simulation phase, either gas or liquid.
                             [Default: gas]
 --size=STEPSIZE             Step size chosen by the user for the C4 value
                             during parameter searching. [Default: 10.0]
 --method=MINM               Optimization method of the C4 terms, The options
                             are: powell, cg, bfgs or slsqp.[Default: bfgs]
                             Please check the website:
                             http://docs.scipy.org/doc/scipy/reference/optimize.html#module-scipy.
                             for more information if interested.
 --platform=PLATF            Platform used. The options are: reference, cpu,
                             gpu or opencl. [Default: cpu] Here we use the
                             OpenMM software to perform the structure
                             minimization. Please check OpenMM user guide for
                             more information if interested.
 --presn=PRESN               Precision used. The options are: single, mixed, or
                             double. This option is only valid when using the CUDA
                             or OpenCL platform. [Default: single]
 --model=MODEL               The metal ion complex model chosen to calculate the sum
                             of unsigned average errors of bond lengths, angles, and
                             dihedrals (the units of them are angstrom, degree and
                             degree respectively while the weights of them are
                             1/100, 1/2 and 1 respectively). This sum is the
                             criterion for the optimization (with a smaller value,
                             the better the parameters). The options are: 1 or 2.
                             1 means a small model (only contains the metal ion
                             and binding heavy atoms) while 2 means a big model
                             (contains the metal ion and heavy atoms in the
                             ligating residues). [Default: 1]
```

15.8.2.5. CartHess2FC.py

```
Usage: CartHess2FC.py -p PDB_file -f QM_output_file [-v software] [-m method]
                      [--scalef freq_scale_factor] [--nstpdb]
                      [--bavg] [--avg13] [--aavg]
Options:
  -h, --help       show this help message and exit
  -i INPUTFILE     Input PDB file name
```

```
-f HESSF          Quantum output file name (a fchk/log file for Gaussian or a file for
                  log file for GAMESS-US).
-v SOFTV          Software version [Default is g03 (means Gaussian03), other
                  options are, g09 (means Gaussian09), and gms (means GAMESS-
                  US)]
-m METHOD         Method used. [Default is sem (means Seminario, applicable
                  for g03, g09, and gms) other option is zmx (means Z-matrix,
                  only applicable for g03 and g09.)]
--scalef=SCALEF   Scale factor (ATTENTION: This is the scale factor of
                  frequency. The force constants will be scaled by
                  multiplying the square of scale_factor).
--nstdpdb         Non standard PDB file used. It is the PDB file which have
                  all the atom names as element followed by interger number.
                  It could be a PDB file generated by software such as
                  antechamber based on the Gaussian output file.
--bavg            Make average of bond force constants based on different ways
                  of chosing sub Hessian matrices using Seminario method.
--avg13           Make average of Urey-Bradley force constants based on
                  different ways of chosing sub Hessian matrices using
                  Seminario method.
--aavg            Make average of angle force constants based on different ways
                  of chosing sub Hessian matrices using Seminario method.
```

CartHess2FF.py is designed to generate force field parameters for bond, angle, Urey-Bradley (1-3 interaction), dihedral, and improper torsion terms based on quantum calculated Cartesian Hessian matrix. These terms could be used seperately in force fields such as AMBER, CHARMM, CNS (Crystallography and NMR System), etc. while limits may be applicable for the dihedral and improper terms (see below).

$$\begin{aligned} E_{total} &= \sum_{bonds} k_b (r - r_0)^2 \\ &+ \sum_{angles} k_\theta (\theta - \theta_0)^2 \\ &+ \sum_{dihedrals} k_\phi (\phi - \phi_0)^2 \end{aligned} \tag{15.1}$$

CartHess2FF.py could generate the force field parameters for the potential shown in Eq15.1 based on the log file of force constant calculation using Gaussian software and Z-matrix method. Here the dihedral term uses a harmonic potential other than a Fourier expansion. Users may try to transfer the parameters to a Fourier term based on the relationship: $V_n = 2k_\phi / n^2$ (whileV_n and$n$ are from Eq14.1) while there is no guarantee for working. This is due to 1-4 nonbonded interaction is usually coupled to dihedral potential in AMBER while it is not considered current potential formulation. Meanwhile, there is also other issues (such as the connectivities of the central two atoms) available which limits the transferability of these dihedral parameters. Therefore only qualitatively comparison between different dihedral angles (while the QM calculations should be carried out under same level of theory) are suggested by the author. The Gaussian calculation could be performed with iop(7/33=1), before which the structural optimization at the same level of theory is needed.

$$
\begin{aligned}
E_{total} &= \sum_{bonds} k_b (r - r_0)^2 \\
&\quad + \sum_{\text{Urey-Bradley}} k_u \left(u - u_0\right)^2 \\
&+ \sum_{angles} k_\theta (\theta - \theta_0)^2 \\
&+ \sum_{dihedrals} k_\phi (\phi - \phi_0)^2 \\
&+ \sum_{impropers} k_{AN} (r_{AN} - r_{AN,0})^2
\end{aligned}
\tag{15.2}
$$

Moreover, CartHess2FF.py could generate the force field parameters for the potential shown in Eq15.2 based on the Cartesian Hessian matrix obtained by using Gaussian or GAMESS-US. Comparing to Eq15.1 it has an additional Urey-Bradley (1-3 harmonic interaction) term and a harmonic improper torsion term (instead of a Fourier term). In the harmonic improper term A and N represent the central atom and its projection into the plane of the other three atoms in the improper torsion. Herein k_{AN} , r_{AN} , and $r_{AN,0}$ represent the force constant, distance and equilibrium distance between A and N respectively. Similarly, parameters of the dihedral and improper torsion terms have limited transferabilty and are only suggested to be used for qualitatively comparison. A fchk file is needed for Gaussian and it could be obtained by using "formchk" command to get it based on the chk file after the force constant calculation (again, before which a structural optimization at the same level of theory is needed). A log file is needed for GAMESS-US force constant calculation (same as Gaussian, a structural optimization at the same level of theory is needed before the calculation).

15.8.2.6. espgen.py

```
Usage: espgen.py -i input_file -o output_file [-v software]
Options:
  -h, --help      show this help message and exit
  -i INPUTFILE    Input file name
  -o OUTPUTFILE   Output file name
  -v SOFTVERSION  Software version [Default is gau (means Gaussian),
                  other option is gms (means GAMESS-US)]
```

15.8.2.7. ProScrs.py

```
Usage: ProScrs.py -i input_file -p PDB_file
Options:
  -h, --help      show this help message and exit
  -i INPUTFILE    Input file name
  -p PDBFILE       PDB file name
  -s PRE           File name prefix (default: MOL)
  -c CHG           Charge (default: 0)
  --symcrd=SYMCRD  Use symbolic Cartesian coordinates (default: 0)
  --fix=FIX        Fix heavy atoms or not (default: 0): 0 means no, 1 means
                   only backbone N, CA, C, and O atoms, 2 means backbone N,
                   CA, C, O, and sidechain beta atoms, 3 means all heavy
                   atoms.
  --crd0=CRD0      Reassign the coordinates with first atom as 0 (default: 0)
```

Reminder: For the following functions, some X-H distances will be adjusted based on the normal X-H distances for the generated Gaussian input files (here X represents a heavy atom).

ace Specify the residue number for which residue you want to treat as ACE (CH3CO). ProScrs.py will keep the backbone CA, HA, C, O atoms, and change beta atom in the sidechain and backbone N atoms into hydrogen atoms while omit all the atoms in the residue.

act Specify the residue number for which residue you want to treat as the CH3CO2- group. This is specific for the C-terminal residue which has backbone O atom coordinated to another atom (e.g. a metal ion). ProScrs.py will keep the backbone CA, HA, C, O, OXT atoms, and change beta atom in the sidechain and backbone N atoms into hydrogen atoms while omit all the atoms in the residue.

nme Specify the residue number for which residue you want to treat as NME (CH3NH). ProScrs.py will keep the backbone CA, HA, N, H atoms, and change beta atom in the sidechain and backbone C atoms into hydrogen atoms while omit all the atoms in the residue.

ant Similar to nme variable except keep all the backbone N atom and H atoms connected to it. This can be a N-terminal residue which has backbone N atom coordinated to another atom (e.g. a metal ion). ProScrs.py will keep the backbone CA, HA, N, H1, H2, H3 atoms, and change beta atom in the sidechain and backbone N atoms into hydrogen atoms while omit all the atoms in the residue.

gly Specify the residue number for which residue you want to treat as the GLY residue. This can be used for the situation that backbone atoms of the residue matters most while sidechain doesn't involve a lot. ProScrs.py will keep the backbone N, H, CA, HA, C, O atoms and change beta atom in the sidechain into a hydrogen atom while omit all the atoms in the residue.

keep Specify the residue number for which residue you want to keep entirely. This can be any residue (such as a metal, water, ligand, or amino acid).

sc Specify the residue number for which residue you only want to keep the sidechain. ProScrs.py will keep the sidechain and backbone CA, HA atoms and change the backbone N and C atoms into hydrogen atoms while omitting all the other backbone atoms.

sc_knh Specify the residue number for which residue you want to keep the sidechain and backbone NH group. ProScrs.py will keep the sidechain and backbone CA, HA, N, H atoms and change the backbone C atom into a hydrogen atom while omitting the O backbone atom.

sc_kco Specify the residue number for which residue you want to keep the sidechain and backbone CO group. ProScrs.py will keep the sidechain and backbone CA, HA, C, O atoms and change the backbone N atom into a hydrogen atom while omitting the H backbone atom.

c2h Specify the residue number for which residue you only want to keep the backbone C atom and change it to a hydrogen atom. This is used to have a hydrogen cap for next residue connecting to it.

n2h Specify the residue number for which residue you only want to keep the backbone N atom and change it to a hydrogen atom. This is used to have a hydrogen cap for former residue connecting to it.

15.8.2.8. car_to_files.py

```
Usage: car_to_files.py -i input_file -m mol2_file -p pdb_file -r residue_name
Options:
  -h, --help       show this help message and exit
  -i INPUT_FILE   Input file name
  -m MOL2_FILE    Output mol2 file name
  -p PDB_FILE     Output PDB file name
  -r RESNAME      Residue name
```

16. Setting up crystal simulations

David S. Cerutti

Simulations of biomolecular crystals are in principle no different than any of the simulations that AMBER does in periodic boundary conditions. However, the setup of these systems is not trivial and probably cannot be accomplished with the LEaP software. Of principal importance are the construction of the solvent conditions (packing precise amounts of multiple solvent species into the simulation cell), and tailoring the unit cell dimensions to accommodate the inherently periodic nature of the system. The LEaP software, designed to construct simulations of molecules in solution, will overlay a pre-equilibrated solvent mask over the (biomolecular) solute, tile that mask throughout the simulation cell, and then prune solvent residues which clash with the solute. The result of this procedure is a system which will likely contract under constant pressure dynamics as the pruning process has left vacuum bubbles at the solute:solvent interface. Simulations of biomolecular crystals require that the simulation cell begin at a size corresponding to the crystallographic unit cell, and deviate very little from that size over the course of equilibration and onset of constant pressure dynamics. This demands a different strategy for placing solvent in the simulation cell. Four programs in the *AmberTools* release are designed to accomplish this. An example of their use in given in a web-based tutorial at *http://ambermd.org/tutorials/advanced/tutorial13/XtalTutor1.html* .

For brevity, only basic descriptions of the programs are given in this manual. All of the programs may be run with command line input; the input options to each program may be listed by running each program with no arguments.

16.1. UnitCell

A macromolecular crystal contains many repeating unit cells which stack like blocks in three dimensional space just as simulation cells do in periodic boundary conditions. Each unit cell, in turn, may contain multiple symmetry-related clusters of atoms. A PDB file contains one set of coordinates for the irreducible unit of the crystal, the "asymmetric unit," and also information about the crystal space group and unit cell dimensions. The *UnitCell* program reads PDB files, seeking the SMTRY records within the REMARKs to enumerate the rotation and translation operations which may be applied to the coordinates given in the PDB file to reconstuct one complete unit cell.

16.2. PropPDB

Simulations in periodic boundary conditions require a minimum unit cell size: the simulation cell must be able to enclose a sphere of at least the nonbonded direct space cutoff radius plus a small buffer region for nonbonded pairlist updates. Many biomolecular crystal unit cells come in "shoebox" dimensions that may have one very short side; many unit cells are also not rectangular but triclinic, meaning that the size of the largest sphere they can enclose is further reduced. For these reasons, and perhaps to ensure that the rigid symmetry imposed by periodic boundary conditions does not create artifacts (crystallographic unit cells are equivalent when averaged over all time and space, but are not necessarily identical at any given moment), it may be necessary to include multiple unit cells within the simulation cell. This is the purpose of the *PropPDB* program: to propagate a unit cell in one or more directions so that the complete simulation cell meets minimum size requirements.

16.3. AddToBox

The *AddToBox* program handles placement of solvent within a crystal unit cell or supercell (as may be created by PropPDB). As described in the introduction, the basic strategy is to place solvent such that added solvent molecules do not clash with biomolecule solutes, but *may* clash with one another initially. This compromise is necessary because enough solvent must be added to the system to ensure that the correct unit cell dimensions are maintained in the long run, but it is not acceptable to place solvent within the interior of a biomolecule where it might not belong and never escape.

The *AddToBox* program takes a PDB file providing the coordinates of a complete biomolecular unit cell or supercell (argument -c), the dimensions by which that supercell repeats in space (-X, -Y, -Z for the three box edge lengths, and -al, -bt, and -gm for the three unit cell angles), a PDB file describing the solvent residue to add (argument -a), and the number of copies of that solvent molecule to add (argument -na). *AddToBox* inherently assumes that the biomolecular unit cell it is initially presented may contain some amount of solvent already, and according to the AMBER convention of listing macromolecular solute atoms first and solvent last assumes that the first -P atoms in the file are the protein (or biomolecule). *AddToBox* will then color a very fine grid "black" if the grid point is within a certain distance of a biomolecular atom (argument -RP) or other solvent atom (argument -RW); the grid is "white" otherwise (the grid is stored in binary for memory efficiency). *AddToBox* will the make a copy of the solvent residue and randomly rotate and translate it somewhere within the unit cell. If all atoms of the solvent residue land on "white" grid voxels, the solvent molecule will become part of the system and the grid around the newly added solvent will be blacked out accordingly. If the solvent molecule cannot be placed, this process will be repeated until a million consecutive failures are encountered, at which point the program will terminate. If *AddToBox* has not placed the requested number of solvent molecules by the time it terminates, the -V option can be used to order the program to recursively call itself with progressively smaller solvent buffer distances until all the requested solvent can be placed. The output of the *AddToBox* program is another PDB named by the -o option.

Successful operation of *AddToBox* may take practice. If multiple solvent species are required, as is the case with heterogenous crystallization solutions, *AddToBox* may be called repeatedly with each input molecular cell being the previous call's output. When considering crystal solvation, the order of addition is important! It is recommended that rare species, such as trace buffer reagents, be added first, with large -RW argument to ensure that they are dispersed throughout the available crystal void zones. Large solvent species such as MPD (an isohexane diol commonly used in crystallization conditions) or should be added second, and with a sufficiently large -RW argument that methyl groups and ring systems cannot become interlocked (which will likely lead to SHAKE / vlimit errors). Small and abundant species such as water should be added last, as they can go anywhere that space remains.

It is likely that the unobservable "void" regions between biomolecules in most crystals *do not* contain solvent species in proportion to their abundance in the crystallization solution–the vast majority of these regions are within a few Ångstroms of some biomolecular surface, and different biomolecular functional groups will preferentially interact with some types of solvent over others. Also, in many crystals some solvent molecules *are* observed; in many of these, the amount of solvent observed is such that it would be impossible to pack other species into the unit cell in proportion to their abundances in the crystallization fluid. In these cases, we recommend estimating the amount of volume that must be filled with solvent *apart from solvent which has already been observed in the crystal*, and filling this void with solvent in proportion to the composition of the crystallization fluid. For example, if a crystal were grown in a 1:1 mole-to-mole water/ethanol mixture, and the crystal coordinates as deposited in the PDB contained 500 water molecules and 3 ethanol molecules, we would use *AddToBox* to add water and ethanol in a 1:1 ratio until the system contained enough solvent to maintain the correct volume during equilibrium dynamics at constant pressure.

Finally, it is difficult to estimate exactly how much solvent will be needed to maintain the correct equilibrium volume; the advisable approach is simply to make an initial guess and script the setup so that, over multiple runs and reconstructions, the correct system composition can be found. We recommend matching the equilibrium unit cell volume to within 0.3% to keep this simulation parameter within the error of most crystallographic measurements. While errors of 0.5-1% will show up quickly after constant pressure dynamics begin, a 10 to 20ns simulation may be needed to ensure that the correct equilibrium volume has been achieved.

16.4. ChBox

After the complex process of adding solvent, the LEaP program may be used to produce a topology and initial set of coordinates based on the PDB file produced by *AddToBox*. By using the *SetBox* command, LEaP will create a periodic system without adding any more solvent on its own. The only problem with using LEaP at this point is that the program will fail to realize that the system *does* tile in three dimensions if only the box dimensions are set properly. If visualized, the output of UnitCell / PropPDB will likely look jagged, but the output of *AddToBox*, containing lots of added water, will make it obvious how parts of biomolecules jutting out one face of the box fit neatly into open spaces on an opposite face. The topology produced by LEaP needs no editing; only the last line of the coordinates does. This can be done manually, but the *ChBox* program automates the process, taking the same coordinates supplied to AddToBox and grafting them into the input coordinates file.

Part IV.

Running simulations

17. sander

17.1. Introduction

This is a guide to *sander*, an Amber module which carries out energy minimization, molecular dynamics, and NMR refinements. The acronym stands for **S**imulated **A**nnealing with **N**MR-**D**erived **E**nergy **R**estraints, but this module is used for a variety of simulations that have nothing to do with NMR refinement. Some of the functionality of sander is available with better computational performance in the pmemd module. In general, sander and pmemd are input compatible. sander inputs for features not supported by pmemd should be properly parsed by pmemd and pmemd should report that the requested feature is not supported. There are a few features available in pmemd that are not supported by sander, see Sections18.3 and 18.4 Some general features are outlined in the following paragraphs:

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

   $$\begin{aligned} V(\mathbf{r}) &= \sum_{bonds} K_b(b-b_0)^2 + \sum_{angles} K_\theta(\theta-\theta_o)^2 \\ &+ \sum_{dihedrals} (V_n/2)(1+\cos[n\phi-\delta] \\ &+ \sum_{nonbij} (A_{ij}/r_{ij}^{12}) - (B_{ij}/r_{ij}^{6}) + (q_i q_j/r_{ij}) \end{aligned}$$

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

   $$E_{pol} = -2\sum_i \mu_i \cdot \mathbf{E}_{io}$$

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

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

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

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

$$\Delta G_{sol} = \sum_{ij}(1-\frac{1}{\varepsilon})(q_i q_j / f_{GB}(r_{ij}) + A\sum_{i}\sigma_i$$

The first term accounts for the polar part of solvation (free) energy, designed to provide an approximation for the reaction field potential, and the second represents the non-polar contribution which is taken to be proportional to the surface area of the molecule.

5. Users can define internal restraints on bonds, valence angles, and torsions, and the force constants and target values for the restraints can vary during the simulation. The relative weights of various terms in the force field can be varied over time, allowing one to implement a variety of simulated annealing protocols in a single run.
6. Internal restraints can be defined to be "time-averaged", that is, restraint forces are applied based on the averaged value of an internal coordinate over the course of the dynamics trajectory, not only on its current value. Alternatively, restraints can be "ensemble-averaged" using the locally-enhanced-sampling (LES) option.
7. Restraints can be directly defined in terms of NOESY intensities (calculated with a relaxation matrix technique), residual dipolar couplings, scalar coupling constants and proton chemical shifts. There are provisions for handling overlapping peaks or ambiguous assignments. In conjunction with distance and angle constraints, this provides a powerful and flexible approach to NMR structural refinements.
8. Replica exchange calculations can allow simultaneous sampling at a variety of conditions (such as temperature), and allow the user to construct Boltzmann samples in ways that converge more quickly than standard MD simulations. Other variants of biased MD simulations can also be used to improve sampling.
9. Restraints can also be defined in terms of the root-mean-square coordinate distance from some reference structure. This allows one to bias trajectories either towards or away from some target. Free energies can be estimated from non-equilibrium simulations based on targetting restraints.
10. Free energy calculations, using thermodynamic integration (TI) with a linear or non-linear mixing of the "unperturbed" and "perturbed" Hamiltonian, can be carried out. Alternatively, potentials of mean force can be computed using umbrella sampling.
11. The empirical valence bond (EVB) scheme can be used to mix "diabatic" states into a potential that can represent many types of chemical reactions that take place in enzymes.
12. QMMM Calculations where part of the system can be treated quantum mechanically allowing bond breaking and formation during a simulation. Semi-empirical and DFTB Hamiltonians are provided directly within *sander*. More advanced *ab initio* and DFT Hamiltonians are available via an interface to external QM software packages.
13. Nuclear quantum effects can be included through path-integral molecular dynamics (PIMD) simulations, and estimates of quantum time-correlation functions can be computed.

17.2. File usage

```
sander [-help] [-O] [-A] -i mdin -o mdout -p prmtop -c inpcrd -r restrt
-ref refc -mtmd mtmd -x mdcrd -y inptraj -v mdvel -frc mdfrc -e mden
-inf mdinfo -radii radii -cpin cpin -cpout cpout -cprestrt cprestrt
-evbin evbin -suffix suffix
-O  Overwrite output files if they exist.
-A  Append output files if they exist (used mainly for replica exchange).
```

Here is a brief description of the files referred to above; the first five files are used for every run, whereas the remainder are only used when certain options are chosen.

mdin *input* control data for the min/md run

mdout *output* user readable state info and diagnostics -o stdout will send output to stdout (to the terminal) instead of to a file.

mdinfo *output* latest mdout-format energy info

prmtop *input* molecular topology, force field, periodic box type, atom and residue names

inpcrd *input* initial coordinates and (optionally) velocities and periodic box size

refc *input* (optional) reference coords for position restraints; also used for targeted MD

mtmd *input* (optional) containing list of files and parameters for targeted MD to multiple targets

mdcrd *output* coordinate sets saved over trajectory

inptraj *input* coordinate sets in trajectory format, when imin=5

mdvel *output* velocity sets saved over trajectory

mdfrc *output* force sets saved over trajectory

mden *output* extensive energy data over trajectory (not synchronized with mdcrd or mdvel)

restrt *output* final coordinates, velocity, and box dimensions if any - for restarting run

inpdip *input* polarizable dipole file, when indmeth=3

rstdip *output* polarizable dipole file, when indmeth=3

cpin *input* protonation state definitions

cprestrt protonation state definitions, final protonation states for restart (same format as cpin)

cpout *output* protonation state data saved over trajectory

evbin *input* input for EVB potentials

suffix *output* this string will be added to all unspecified output files that are printed (for *multisander* runs, it will append this suffix to all output files)

17.3. Example input files

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

1. Simple restrained minimization

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

2. "Plain" molecular dynamics run

```
molecular dynamics run
&cntrl
imin=0, irest=1, ntx=5, (restart MD)
ntt=3, temp0=300.0, gamma_ln=5.0, (temperature control)
ntp=1, taup=2.0, (pressure control)
ntb=2, ntc=2, ntf=2, (SHAKE, periodic bc.)
nstlim=500000, (run for 0.5 nsec)
ntwx=1000, ntpr=200, (output frequency)
/
```

3. Self-guided Langevin dynamics run

```
Self-guided Langevin dynamics run
&cntrl
imin=0, irest=0, ntx=1, (start LD)
ntt=3, temp0=300.0,gamma_ln=1.0, (temperature control)
ntc=3, ntf=3, (SHAKE)
nstlim=500000, (run for 0.5 nsec)
ntwx=1000, ntpr=200, (output frequency)
isgld=1, tsgavg=0.2,tempsg=400.0, (SGLD)
/
```

17.4. Namelist Input Syntax

Namelist provides list-directed input, and convenient specification of default values. It dates back to the early 1960's on the IBM 709, but was regrettably not part of Fortran 77. It is a part of the Fortran 90 standard, and is supported as well by most Fortran 77 compilers (including g77).

Namelist input groups take the form:

```
&name
var1=value, var2=value, var3(sub)=value,
var4(sub,sub,sub)=value,value,
var5=repeat*value,value,
/
```

The variables must be names in the Namelist variable list. The order of the variables in the input list is of no significance, except that if a variable is specified more than once, later assignments may overwrite earlier ones. Blanks may occur anywhere in the input, except embedded in constants (other than string constants, where they count as ordinary characters).

It is common in older inputs for the ending "/" to be replaced by "&end"; this is non-standard-conforming.

Letter case is ignored in all character comparisons, but case is preserved in string constants. String constants must be enclosed by single quotes ('). If the text string itself contains single quotes, indicate them by two consecutive single quotes, e.g. C1' becomes 'C1''' as a character string constant.

Array variables may be subscripted or unsubscripted. An unsubscripted array variable is the same as if the subscript (1) had been specified. If a subscript list is given, it must have either one constant, or exactly as many as the number in the declared dimension of the array. Bounds checking is performed for ALL subscript positions, although if only one is given for a multi-dimension array, the check is against the entire array size, not against the first dimension. If more than one constant appears after an array assignment, the values go into successive locations of the array. It is NOT necessary to input all elements of an array.

Any constant may optionally be preceded by a positive (1,2,3,..) integer repeat factor, so that, for example, 25*3.1415 is equivalent to twenty-five successive values 3.1415. The repeat count separator, *, may be preceded and followed by 0 or more blanks. Valid LOGICAL constants are 0, F, .F., .FALSE., 1, T, .T., and .TRUE.; lower case versions of these also work.

17.5. Overview of the information in the input file

General minimization and dynamics input One or more title lines, followed by the (required) &cntrl and (optional) &pb, &ewald, &qmmm, &amoeba or &debugf namelist blocks. Described in Sections 17.6 and 17.7.

Varying conditions Parameters for changing temperature, restraint weights, etc., during the MD run. Each parameter is specified by a separate &wt namelist block, ending with &wt type="END", /. Described in Section 17.8.

File redirection TYPE=*filename* lines. Section ends with the first non-blank line which does not correspond to a recognized redirection. Described in Section 17.9.

Group information Read if *ntr*, *ibelly* or *idecomp* are set to nonzero values, and if some other conditions are satisfied; see sections on these variables, below. Described in Appendix 19.3.

17.6. General minimization and dynamics parameters

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

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

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

17.6.1. General flags describing the calculation

imin Flag to run minimization.

- **= 0** (default) Run molecular dynamics without any minimization.
- **= 1** Perform an energy minimization.
- **= 5** Read in a trajectory for analysis.

  Although sander will write energy information in the output files (using *ntpr*), it is often desirable to calculate the energies of a set of structures at a later point. In particular, one may wish to post-process a set of structures using a different energy function than was used to generate the structures. An example of this is MM-PBSA analysis, where the explicit water is removed and replaced with a continuum model.

  If *imin* is set to 5, sander will read a trajectory file (the “inptraj” argument, specified using `-y` on the command line), and will perform the functions described in the mdin file (e.g., an energy minimization) for each of the structures in this file. The final structure from each minimization will be written out to the normal mdcrd file. If you wish to read in a binary (i.e., NetCDF format) trajectory, be sure to set *ioutfm* to 1 (see below). Note that this will result in the output trajectory having NetCDF format as well.

For example, when *imin* = 5 and *maxcyc* = 1000, sander will minimize each structure in the trajectory for 1000 steps and write a minimized coordinate set for each frame to the mdcrd file. If *maxcyc* = 1, the output file can be used to extract the energies of each of the coordinate sets in the inptraj file.

Trajectories containing box coordinates can be post-processed. In order to read trajectories with box coordinates, *ntb* should be greater than 0.

IMPORTANT CAVEAT: The initial coordinates input file used (`-c` <inpcrd>) should be the same as the initial coordinates input file usedto generate the original trajectory. This is because sander sets up parameters for PME from the box coordinates in the initial coordinates input file.

nmropt **= 0** (default) No nmr-type analysis will be done.

= 1 NMR restraints and weight changes will be read.

= 2 NMR restraints, weight changes, NOESY volumes, chemical shifts and residual dipolar restraints will be read.

17.6.2. Nature and format of the input

ntx Option to read the initial coordinates, velocities and box size from the inpcrd file. Option 1 must be used when one is starting from minimized or model-built coordinates. If an MD restrt file is used as inpcrd, then option 5 is generally used (unless you explicitly wish to ignore the velocities that are present).

= 1 (default) Coordinates, but no velocities, will be read; either formatted (ASCII) files or NetCDF files can be used, as the input file type will be auto-detected.

= 5 Coordinates and velocities will be read from either a NetCDF or a formatted (ASCII) coordinate file. Box information will be read if *ntb* $>$ 0. The velocity information will only be used if *irest* = 1 (see below).

irest Flag to restart a simulation.

= 0 (default) Do not restart the simulation; instead, run as a new simulation. Velocities in the input coordinate file, if any, will be ignored, and the time step count will be set to 0 (unless overridden by *t*; see below).

= 1 Restart the simulation, reading coordinates and velocities from a previously saved restart file. The velocity information is necessary when restarting, so *ntx* (see above) must be 4 or higher if *irest* = 1.

17.6.3. Nature and format of the output

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

= 1 Formatted (ASCII)

= 2 (default) NetCDF file (recommended, unless you have a workflow that requires the formatted form.)

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

ntave	Every *ntave* steps of dynamics, running averages of average energies and fluctuations over the last *ntave* steps will be printed out. A value of 0 disables this printout. Setting *ntave* to a value 1/2 or 1/4 of *nstlim* provides a simple way to look at convergence during the simulation. Default = 0 (disabled).
ntwr	Every *ntwr* steps during dynamics, the “restrt” file will be written, ensuring that recovery from a crash will not be so painful. No matter what the value of *ntwr*, a restrt file will be written at the end of the run, i.e., after *nstlim* steps (for dynamics) or *maxcyc* steps (for minimization). If $ntwr < 0$, a unique copy of the file, “restrt_<nstep>”, is written every abs(*ntwr*) steps. This option is useful if for example one wants to run free energy perturbations from multiple starting points or save a series of restrt files for minimization. Default = nstlim.
iwrap	If *iwrap* = 1, the coordinates written to the restart and trajectory files will be "wrapped" into a primary box. This means that for each molecule, its periodic image closest to the middle of the "primary box" (with x coordinates between 0 and a, y coordinates between 0 and b, and z coordinates between 0 and c) will be the one written to the output file. This often makes the resulting structures look better visually, but has no effect on the energy or forces. Performing such wrapping, however, can mess up diffusion and other calculations. If *iwrap* = 0, no wrapping will be performed, in which case it is typical to use *cpptraj* as a post-processing program to translate molecules back to the primary box. For very long runs, setting *iwrap* = 1 may be required to keep the coordinate output from overflowing the trajectory and restart file formats, especially if trajectories are written in ASCII format instead of NetCDF (see also the *ioutfm* option). Default = 0.
ntwx	Every *ntwx* steps, the coordinates will be written to the mdcrd file. If *ntwx* = 0, no coordinate trajectory file will be written. Default = 0.
ntwv	Every *ntwv* steps, the velocities will be written to the mdvel file. If *ntwv* = 0, no velocity trajectory file will be written. If *ntwv* = -1, velocities will be written to mdcrd, which then becomes a combined coordinate/velocity trajectory file, at the interval defined by *ntwx*. This option is available only for binary NetCDF output (*ioutfm* = 1). Most users will have no need for a velocity trajectory file and so can safely leave *ntwv* at the default. Default = 0. Note that dumping velocities frequently, like forces or coordinates, will introduce potentially significant I/O and communication overhead, hurting both performance and parallel scaling.
ntwf	Every *ntwf* steps, the forces will be written to the mdfrc file. If *ntwf* = 0, no force trajectory file will be written. If *ntwf* = -1, forces will be written to the mdcrd, which then becomes a combind coordinate/force trajectory file, at the interval defined by *ntwx*. This option is available only for binary NetCDF output (*ioutfm* = 1). Most users will have no need for a force trajectory file and so can safely leave *ntwf* at the default. Default = 0. Note that dumping forces frequently, like velocities or coordinates, will introduce potentially significant I/O and communication overhead, hurting both performance and parallel scaling.
ntwe	Every *ntwe* steps, the energies and temperatures will be written to file "mden" in a compact form. If *ntwe* = 0 then no mden file will be written. Note that energies in the mden file are not synchronized with coordinates or velocities in the mdcrd or mdvel file(s). Assuming identical *ntwe* and *ntwx* values the energies are one time step before the coordinates (as well as the velocities which are synchronized with the coordinates). Consequently, an mden file is rarely written. Default = 0.
ioutfm	The format of coordinate and velocity trajectory files (mdcrd, mdvel and inptraj). As of Amber 9, the binary format used in previous versions is no longer supported; binary output is now in NetCDF trajectory format. Binary trajectory files have many advantages: they are smaller, higher precision, much faster to read and write, and able to accept a wider range of coordinate (or velocity) values than formatted trajectory files. **= 0** Formatted ASCII trajectory **= 1** (default) Binary NetCDF trajectory

ntwprt The number of atoms to include in trajectory files (mdcrd and mdvel). This flag can be used to decrease the size of the these files, by including only the first part of the system, which is usually of greater interest (for instance, one might include only the solute and not the solvent). If *ntwprt* = 0, all atoms will be included.

= 0 (default) Include all atoms of the system when writing trajectories.

> 0 Include only atoms 1 to *ntwprt* when writing trajectories.

idecomp Perform energy decomposition according to a chosen scheme. In former distributions this option was only really useful in conjunction with mm_pbsa, where it is turned on automatically if required. Now, a decomposition of $\langle \partial V / \partial \lambda \rangle$ on a per-residue basis in thermodynamic integration (TI) simulations is also possible.[344] The options are:

= 0 (default) Do not decompose energies.

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

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

= 3 Decompose energies on a pairwise per-residue basis; otherwise equivalent to *idecomp* = 1. Not available in TI simulations.

= 4 Decompose energies on a pairwise per-residue basis; otherwise equivalent to *idecomp* = 2. Not available in TI simulations.

If energy decomposition is requested, residues may be chosen by the RRES and/or LRES card. The RES card is used to select the residues about which information is written out. See chapters 21.1 or 31 for more information. Use of *idecomp* > 0 is incompatible with *ntr* > 0 or *ibelly* > 0.

17.6.4. Frozen or restrained atoms

ibelly Flag for belly type dynamics. If set to 1, 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 with *bellymask*. This option is not available when *igb>0*. When belly type dynamics is in use, bonded energy terms, vdW interactions, and direct space electrostatic interactions are *not calculated* for pairs of frozen atoms. Note that this does *not* provide any significant speed advantage. Freezing atoms can be useful for some applications but is maintained primarily for backwards compatibility with older versions of Amber. Most applications should use the *ntr* variable instead to restrain parts of the system to stay close to some initial configuration. Default = 0.

ntr Flag for restraining specified atoms in Cartesian space using a harmonic potential, if ntr > 0. The restrained atoms are determined by the *restraintmask* string. The force constant is given by *restraint_wt*. The coordinates are read in "restrt" format from the "refc" file. Default = 0.

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

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

bellymask String that specifies the *moving* atoms when *ibelly=1*.

The syntax for both *restraintmask* and *bellymask* is given in Section 19.1.1. Note that these mask strings are limited to a maximum of 256 characters.

17.6.5. Energy minimization

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

ntmin Flag for the method of minimization.

- **= 0** Full conjugate gradient minimization. The first 4 cycles are steepest descent at the start of the run and after every nonbonded pairlist update.
- **= 1** For NCYC cycles the steepest descent method is used then conjugate gradient is switched on (default).
- **= 2** Only the steepest descent method is used.
- **= 3** The XMIN method is used, see Section 20.7.1.
- **= 4** The LMOD method is used, see Section 20.7.2.

dx0 The initial step length. If the initial step length is too big then will give a huge energy; however the minimizer is smart enough to adjust itself. Default 0.01.

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

17.6.6. Molecular dynamics

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

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

For Langevin dynamics, the *position* of the center-of-mass of the molecule is reset to zero every NSCM steps, but the velocities are not affected. Hence there is no change to either the translation or rotational components of the momenta. (Doing anything else would destroy the way in which temperature is regulated in a Langevin dynamics system.) The only reason to even reset the coordinates is to prevent the molecule from diffusing so far away from the origin that its coordinates overflow the format used in restart or trajectory files.

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

dt The time step (psec). Recommended MAXIMUM is .002 if SHAKE is used, or .001 if it isn't. Note that for temperatures above 300K, the step size should be reduced since greater temperatures mean increased velocities and longer distance traveled between each force evaluation, which can lead to anomalously high energies and system blowup. Default 0.001.
The use of Hydrogen Mass Repartitioning (HMR) (see [324] and references therein for more information), together with SHAKE, allows the time step to be increased in a stable fashion by about a factor of two (up to .004) by slowing down the high frequency hydrogen motion in the system. To use HMR, the masses in the topology file need to be altered before starting the simulation. ParmEd can do this automatically with the HMassRepartition option; see Section 14.2 .

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

17.6.7. Temperature regulation

ntt Switch for temperature scaling. Note that setting *ntt*=0 corresponds to the microcanonical (NVE) ensemble (which should approach the canonical one for large numbers of degrees of freedom). Some aspects of the "weak-coupling ensemble" (*ntt*=1) have been examined, and roughly interpolate between the microcanonical and canonical ensembles.[345, 346] The *ntt*=2 and 3 options correspond to the canonical (constant T) ensemble.

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

= 1 Constant temperature, using the weak-coupling algorithm.[347] A single scaling factor is used for all atoms. Note that this algorithm just ensures that the total kinetic energy is appropriate for the desired temperature; it does nothing to ensure that the temperature is even over all parts of the molecule. Atomic collisions will tend to ensure an even temperature distribution, but this is not guaranteed, and there are many subtle problems that can arise with weak temperature coupling.[348] Using *ntt=1* is especially dangerous for generalized Born simulations, where there are no collisions with solvent to aid in thermalization.) Other temperature coupling options (especially *ntt*=3) should be used instead.

= 2 Andersen-like temperature coupling scheme,[349] in which imaginary "collisions" randomize the velocities to a distribution corresponding to *temp0* every *vrand* steps. Note that in between these "massive collisions", the dynamics is Newtonian. Hence, time correlation functions (etc.) can be computed in these sections, and the results averaged over an initial canonical distribution. Note also that too high a collision rate (too small a value of *vrand*) will slow down the speed at which the molecules explore configuration space, whereas too low a rate means that the canonical distribution of energies will be sampled slowly. A discussion of this rate is given by Andersen.[350] Note that this option is not equivalent to the original thermostat described by Andersen[350].

= 3 Use Langevin dynamics with the collision frequency γ given by *gamma_ln*, discussed below. Note that when γ has its default value of zero, this is the same as setting *ntt* = 0. Since Langevin simulations are highly susceptible to "synchronization" artifacts,[351, 352] you should explicitly set the *ig* variable (described below) to a different value at each restart of a given simulation.

= 9 Optimized Isokinetic Nose-Hoover chain ensemble (OIN) [240, 353]. Constant temperature simulation utilizing Nose-Hoover chains and an isokinetic constraint on the particle and thermostat velocities, implemented for use in multiple time-stepping methods, namely for 3D-RISM and RESPA. Stabilizes and smooths particle dynamics and mitigates resonance instabilities, allowing for larger intermediate times steps, up to 16 fs for RESPA (*nrespa*=16 for *dt*=0.001) and 8 fs for 3D-RISM MTS size (*rismnrespa*=8). Each atom is coupled to three Nose-Hoover chains per atom and the thermostat coupling constant (relaxation time) is determined from 1/*gamma_ln*, hence *gamma_ln* must be > 0 if *ntt*=9 invoked. Variable *nkija* specifies the number of substeps of *dt* to use for integrating the equations of motion and *idistr* specifies the frequency at which the thermostat velocity distribution functions are accumulated (if > 0). Such functions are written at frequency *ntpr*. Two additional files containing the thermostat and chain restart velocities, *tfreeze.rst* and *vfreez.rst*, are written at frequency *ntwr*.

= 10 Stochastic Isokinetic Nose-Hoover RESPA integrator [354]. A novel isokinetic integrator developed by Tuckerman and co-workers that invokes an isokinetic constraint on the particle velocities combined with *nkija* (see below) auxiliary thermostat velocities *v1* and *v2*. The integrator includes a stochastic component in the equations of motion, which introduces white noise into the system, for the purpose of minimizing resonance instabilities in the velocities, ultimately allowing for larger RESPA steps. The isokinetic constraint has the form $mv^2 + \frac{L}{L+1}\sum_{i=1}^{L} Q_1 v_{1i}^2 = Lk_BT$. Here L is the number of additional thermostat degrees of freedom, defined in AMBER as *nkija* (see below), and $Q_1$is the thermostat mass, determined from *sinrtau* (below), v is the particle velocity and v_1 is one of two auxiliary velocities (e.g. thermostat velocities), and m, k_B, and T, are the particle mass, Boltzmann constant, and system temperature (*temp0*), respectively. In using this integrator, the system is placed in the isokinetic ensemble, as such the velocities are NOT canonical and no thermodynamic observables can be derived from them. This will lead to anomalous temperature readings throughout the simulation - for 1 thermostat degree of freedom (L = *nkija* = 1) the temperature will appear about one-half the specified temperature (*temp0*), and with additional thermostat DOF, the temperature will approach, but never exceed, the desired temperature, *temp0.* However, the particle coordinates ARE canonical and it can be said the configurations obtained from a simulation were sampled from a Boltzmann distribution at the specified temperature (*temp0*).

temp0	Reference temperature at which the system is to be kept, if *ntt* $>$ 0. Note that for temperatures above 300K, the step size should be reduced since increased distance traveled between evaluations can lead to SHAKE and other problems. Default 300.
temp0les	This is the target temperature for all LES particles (see Chapter 6). If *temp0les*$<$0, a single temperature bath is used for all atoms, otherwise separate thermostats are used for LES and non-LES particles. Default is -1, corresponding to a single (weak-coupling) temperature bath.
tempi	Initial temperature. For the initial dynamics run, (NTX .lt. 3) the velocities are assigned from a Maxwellian distribution at TEMPI K. If TEMPI = 0.0, the velocities will be calculated from the forces instead. TEMPI has no effect if NTX .gt. 3. Default 0.0.
ig	The seed for the pseudo-random number generator. The MD starting velocity is dependent on the random number generator seed if NTX .lt. 3 .and. TEMPI .ne. 0.0. The value of this seed also affects the set of pseudo-random values used for Langevin dynamics or Andersen-like coupling, and hence should be set to a different value on each restart if *ntt* = 2 or 3. If *ig=-1*, (the default) the random seed will be based on the current date and time, and hence will be different for every run. It is recommended that, unless you specifically desire reproducibility, that you set ig=-1 for all runs involving *ntt*=2 or 3.
tautp	Time constant, in ps, for heat bath coupling for the system, if *ntt* = 1. Default is 1.0. Generally, values for TAUTP should be in the range of 0.5-5.0 ps, with a smaller value providing tighter coupling to the heat bath and, thus, faster heating and a less natural trajectory. Smaller values of TAUTP result in smaller fluctuations in kinetic energy, but larger fluctuations in the total energy. Values much larger than the length of the simulation result in a return to constant energy conditions.
gamma_ln	The collision frequency γ, in ps^{-1}, when *ntt* = 3. Default is 0. A simple Leapfrog integrator is used to propagate the dynamics, with the kinetic energy adjusted to be correct for the harmonic oscillator case.[355, 356] Note that it is not necessary that γ approximate the physical collision frequency, which is about 50 ps^{-1} for liquid water. In fact, it is often advantageous, in terms of sampling[356, 357] or stability of integration[358], to use much smaller values, around 2 to 5 ps^{-1}.[356, 358] For implicit solvent (GB), even much lower values may be useful: for example, setting gamma_ln to 0.01 ps^{-1} can lead to significant, up to 100-fold in some cases, speedup of conformational sampling.[125] Also used to determine thermostat coupling constant for the Optimized Isokinetic Nose-Hoover chain integrator (OIN, ntt=9), which is equal to 1/*gamma_ln* [240], so the specified *gamma_ln* must be > 0. A *gamma_ln* of 10 ps^{-1} represents a coupling constant of 100 fs. For *ntt=10*, this is the friction constant associated with the stochastic component of the integrator, essentially serving the same role as in the langevein integrator [354]. This parameter is required for *ntt=10* and must be > 0.
vrand	If *vrand>0* and *ntt=2*, the velocities will be randomized to temperature TEMP0 every *vrand* steps. Default is 1000.

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

nkija For use with *ntt*=9 and *ntt=10*., For *ntt=9*, this the number of substeps of *dt* when integrating the thermostat equations of motion, for greater accuracy. For *ntt=10*, this specifies the number of additional auxiliary velocity variables *v1* and *v2*, which will total nkija$\times v1$ +nkija$\times v2$ *[354]*. Default is 1 for both integrators.

idistr For the isokinetic integrator (*ntt*=9), the frequency at which the thermostat velocity distribution functions are accumulated.

sinrtau For the SINR (Stochastic Isokinetic Nose-Hoover RESPA) integrator (*ntt=10*), this specifies the time scale for determining the masses associated with the two auxiliary velocity variables *v1* and *v2* (e.g. thermostat velocities) and hence the magnitude of the coupling of the physical velocities with the auxiliary velocities. Generally this should be related to the time scale of the system. See [354] for more explanation. Default is 1.0.

17.6.8. Pressure regulation

In "constant pressure" dynamics, the volume of the unit cell is adjusted (by small amounts on each step) to make the computed pressure approach the target pressure, *pres0*. Equilibration with $ntp > 0$ is generally necessary to adjust the density of the system to appropriate values. Note that fluctuations in the instantaneous pressure on each step will appear to be large (several hundred bar), but the average value over many steps should be close to the target pressure. Pressure regulation only applies when Constant Pressure periodic boundary conditions are used (*ntp* > *0*). The two available pressure coupling algorithms available in Amber are of the “weak-coupling” variety, analogous to temperature coupling,[347] and the use of the Monte Carlo barostat. While the Berendsen barostat yields the correct target density, it does not strictly sample from the isothermal-isobaric ensemble and typically yields volume fluctuations that are too low. The Monte Carlo barostat, on the other hand, samples rigorously from the isobaric-isothermal ensemble and does not necessitate computing the virial. Please note: in general you will need to equilibrate the temperature to something like the final temperature using constant volume (*ntp*=0) *before* switching on constant pressure simulations to adjust the system to the correct density. If you fail to do this, the program will try to adjust the density too quickly, and bad things (such as SHAKE failures) are likely to happen.

ntp Flag for constant pressure dynamics. This option should be set to 1 or 2 when Constant Pressure periodic boundary conditions are used.

= 0 No pressure scaling (Default)

= 1 md with isotropic position scaling

= 2 md with anisotropic (x-,y-,z-) pressure scaling: this should only be used with orthogonal boxes (i.e. with all angles set to 90 degrees). Anisotropic scaling is primarily intended for non-isotropic systems, such as membrane simulations, where the surface tensions are different in different directions; it is generally not appropriate for solutes dissolved in water.)[240]

= 3 md with semiisotropic pressure scaling: this is only available with constant surface tension (csurften > 0) and orthogonal boxes. This links the pressure coupling in the two directions tangential to the interface.

barostat Flag used to control which barostat to use in order to control the pressure.

= 1 Berendsen (Default)

= 2 Monte Carlo barostat

mcbarint Number of steps between volume change attempts performed as part of the Monte Carlo barostat. Default is 100.

pres0 Reference pressure (in units of bars, where 1 bar $\approx$ 0.987 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 1.0×10^{-6} bar^{-1}; a value of 44.6 (default) is appropriate for water.

taup Pressure relaxation time (in ps), when NTP $>$ 0. The recommended value is between 1.0 and 5.0 psec. Default value is 1.0, but larger values may sometimes be necessary (if your trajec)[240]tories seem unstable).

Surface tension regulation

Constant surface tension is used in statistical ensembles for simulating liquid interfaces. This is primarily intended for lipid membrane simulations with two or more interfaces. Constant surface tension is only available for simulations with anisotropic pressure or semiisotropic scaling. This algorithm is an extension to the Berendsen pressure scaling algorithm that adjusts the tangential pressure evaluation in order to maintain a “constant” surface tension.[359] Since the surface tension is a function of the pressure tensor, fluctuations of the surface tension will be large.

In order to use constant surface tension, periodic boundary conditions (ntb = 2), anisotropic or semiisotropic pressure scaling (ntp = 2 or ntp =3), and an orthogonal box must be used.

csurften Flag for constant surface tension dynamics.

= 0 No constant surface tension (default)

= 1 Constant surface tension with interfaces in the yz plane

= 2 Constant surface tension with interfaces in the xz plane

= 3 Constant surface tension with interfaces in the xy plane

gamma_ten Surface tension value in units of dyne/cm. Default value is 0.0 dyne/cm.

ninterface Number of interfaces in the periodic box. There must be at least two interfaces in the periodic box. Two interfaces is appropriate for a lipid bilayer system and is the default value.

17.6.9. SHAKE bond length constraints

ntc Flag for SHAKE to perform bond length constraints.[360] (See also NTF in the **Potential function** section. In particular, typically NTF = NTC.) The SHAKE option should be used for most MD calculations. The size of the MD timestep is determined by the fastest motions in the system. SHAKE removes the bond stretching freedom, which is the fastest motion, and consequently allows a larger timestep to be used. For water models, a special "three-point" algorithm is used.[361] Consequently, to employ TIP3P set NTF = NTC = 2.

Since SHAKE is an algorithm based on dynamics, the minimizer is not aware of what SHAKE is doing; for this reason, minimizations generally should be carried out without SHAKE. One exception is short minimizations whose purpose is to remove bad contacts before dynamics can begin.

For parallel versions of *sander* only intramolecular atoms can be constrained. Thus, such atoms must be in the same chain of the originating PDB file.

= 1 SHAKE is not performed (default)

= 2 bonds involving hydrogen are constrained

= 3 all bonds are constrained (not available for parallel or qmmm runs in *sander*) tol Relative geometrical tolerance for coordinate resetting in shake. Recommended maximum: <0.00005 Angstrom Default 0.00001.

jfastw Fast water definition flag. By default, the system is searched for water residues, and special routines are used to SHAKE these systems.[361]

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

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

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

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

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

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

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

noshakemask String that specifies atoms that are not to be shaken (assuming that ntc>1). Any bond that would otherwise be shaken by virtue of the *ntc* flag, but which involves an atom flagged here, will *not* be shaken. The syntax for this string is given in Chap. 13.5. Default is an empty string, which matches nothing. A typical use would be to remove SHAKE constraints from all or part of a solute, while still shaking rigid water models like TIPnP or SPC/E. Another use would be to turn off SHAKE constraints for the parts of the system that are being changed with thermodynamic integration, or which are the EVB or quantum regions of the system.

If this option is invoked, then all parts of the potential must be evaluated, that is, *ntf* must be one. The code enforces this by setting *ntf* to 1 when a *noshakemask* string is present in the input.

If you want the *noshakemask* to apply to all or part of the water molecules, you must also set *jfastw=4*, to turn off the special code for water SHAKE. (If you are not shaking waters, you presumably also want to issue the "set default FlexibleWater on" command in LEaP; see that chapter for more information.)

17.6.10. Water cap

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

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

= 1 With this option, a cap can be excised from a larger box of water. For this, cutcap (i.e., the radius of the cap), xcap, ycap, and zcap (i.e., the location of the center of the cap) need to be specified in the &cntrl namelist. Note that the cap parameters must be chosen such that the whole solute is covered by solvent. Solvent molecules (and counterions) located outside the cap are ignored. Although this option also works for minimization and dynamics calculations in general, it is intended to post-process snapshots in the realm of MM-PBSA to get a linear-response approximation of the solvation free energy, output as ’Protein-solvent interactions’.

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

= 5 With this option, a shell of water around a solute can be excised from a larger box of water. For this, cutcap (i.e., the thickness of the shell) needs to be specified in the &cntrl namelist. Solvent molecules (and counterions) located outside the cap are ignored. This option only works for a single-step minimization. It is intended to post-process snapshots in the realm of MM-PBSA to get a linear-response approximation of the solvation free energy, output as ’Protein-solvent interactions’.

fcap The force constant for the cap restraint potential.

cutcap Radius of the cap, if *ivcap=1* is used.

xcap,ycap,zcap Location of the cap center, if *ivcap=1* is used.

17.6.11. NMR refinement options

(Users to should consult the section NMR refinement to see the context of how the following parameters would be used.)

iscale Number of additional variables to optimize beyond the 3N structural parameters. (Default = 0). At present, this is only used with residual dipolar coupling and CSA or pseudo-CSA restraints.

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 This parameter determines the the functional form of the penalty function for NOESY volume and chemical shift restraints.

= 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.

17.6.12. EMAP restraints

EMAP restraints are used to perform targeted conformational search (TCS)[362]. EMAP uses maps to define restraints to maintain conformations and/or to induce simulation systems to the target conformations. The restraint map can be either obtained from electron microscopy experiments or derived from known protein structures, or defined from initial simulation coordinates. EMAP can be used to do rigid docking of molecules into maps and to do flexible fitting to obtain conformations defined by experimental maps. EMAP can also be used to maintain conformations of protein domains when studying large scale conformational change. Users should consult the section 23.13 to see how to define EMAP restraints.

iemap Turn on EMAP restrained simulation when *iemap*>0. (Default = 0). EMAP restraint information must be input from *&emap* namelists in the input file.

gammamap Friction constant for the EMAP restraint maps when allowed to move. (Default=1/ps). (See Section 23.13)

17.7. Potential function parameters

The parameters in this section generally control what sort of force field (or potential function) is used for the simulation.

17.7.1. Generic parameters

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 This variable controls whether or not periodic boundaries are imposed on the system during the calculation of non-bonded interactions. Bonds spanning periodic boundaries are not yet supported. There is no longer any need to set this variable, since it can be determined from igb and ntp parameters. The “proper” default for ntb is chosen (ntb=0 when igb > 0, ntb=2 when ntp > 0, and ntb=1 otherwise). This behavior can be overridden by supplying an explicit value, although this is discouraged to prevent errors. The allowed values for NTB are

= 0 no periodicity is applied and PME is off (default when igb > 0)
= 1 constant volume (default when igb and ntp are both 0, which are their defaults)
= 2 constant pressure (default when ntp > 0)

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

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

dielc Dielectric multiplicative constant for the electrostatic interactions. Default is 1.0. Please note this is NOT related to dielectric constants for generalized Born or Poisson-Boltzmann calculations. It should only be used for quasi-vacuum simulations, e.g. where one wants $\varepsilon = 4r$; in this case you would also set the *eedmeth* variable as well.

cut This is used to specify the nonbonded cutoff, in Angstroms. For PME, the cutoff is used to limit direct space sum, and 8.0 is usually a good value. When *igb>0*, the cutoff is used to truncate nonbonded pairs (on an atom-by-atom basis); here a larger value than the default is generally required. A separate parameter (**RGBMAX**) controls the maximum distance between atom pairs that will be considered in carrying out the pairwise summation involved in calculating the effective Born radii, see the generalized Born section below.
When *igb > 0*, the default is 9999.0 (effectively infinite)
When *igb==0*, the default is 8.0.

fswitch	When off, *fswitch<=0* , uses a truncation cutoff. When on *fswitch>0*, sets a force switching region where the force cutoff smoothly approaches 0 between the region of the fswitch value to the cut value. Force values below the fswitch value follow the standard Lennard-Jones force. Default is -1. This option is not supported for use with GB (i.e., only *igb=0* and *ntb>0*), nor is it compatible with the 12-6-4 Lennard-Jones model (*lj1264=1*). Due to performance regressions (about 20%) with running with the force switching on, it is recommended that simulations run with fswitch off unless using a force field that requires or recommends using the force switch.
nsnb	Determines the frequency of nonbonded list updates when *igb=0* and *nbflag=0*; see the description of *nbflag* for more information. Default is 25.
ipol	When set to 1, use a polarizable force field. See Section 17.7.5 for more information. Default is 0.
ifqnt	Flag for QM/MM run; if set to 1, you must also include a &qmmm namelist. See Section 6.4 for details on this option. Default is 0.
igb	Flag for using the generalized Born or Poisson-Boltzmann implicit solvent models. See Section 4 for information about using this option. Default is 0.
irism	Flag for 3D-reference interaction site model (RISM) molecular solvation method. See Section7.7for information about this option. Default is 0.
ievb	If set to 1, use the empirical valence bond method to compute energies and forces. See Section 6.3 for information about this option. Default is 0.
iamoeba	Flag for using the *amoeba* polarizable potentials of Ren and Ponder.[363, 364] When this option is set to 1, you need to prepare an amoeba namelist with additional parameters. Also, the *prmtop* file is built in a special way. See Section **??** for more information about this option. Default is 0.
lj1264	In general, you should rarely have to set this variable. When the Lennard-Jones C-coefficient is found in your prmtop file, the default value is set to 1 (meaning it is active). When this flag is *not* present in the prmtop file, the default value is set to 0 (meaning the 12-6-4 potential [95] is inactive). Setting this to 0 when the C-coefficient is present will forcibly turn off the 12-6-4 potential. Setting lj1264 to 1 when no C-coefficient is present will result in a fatal error. Therefore, this flag can be used to quickly disable the r^{-4} term. However, the remaining L-J parameters will still be optimized for the 12-6-4 potential, so this should only be done when testing! It currently only supports *sander* and *pmemd* (both the serial and MPI versions) but not *pmemd.cuda*. It is currently only compatible with the Particle Mesh Ewald method for long-range electrostatics. For more information please see Section 3.6. For adding it to your topology file, see Subsection 14.2.2.6.
efx	This sets the x component of the electric field in kcal/(mol*A*e). Electric fields are naturally off if efx, efy, efz are 0. Default value is 0. It currently only supports *pmemd* (both the serial and MPI versions).
efy	This sets the y component of the electric field in kcal/(mol*A*e). Electric fields are naturally off if efx, efy, efz are 0. Default value is 0. It currently only supports *pmemd* (both the serial and MPI versions).
efz	This sets the z component of the electric field in kcal/(mol*A*e). Electric fields are naturally off if efx, efy, efz are 0. Default value is 0. It currently only supports *pmemd* (both the serial and MPI versions).
efn	If efn is on (efn=1), the x, y, z (efx, efy, efz) components are scaled to box size. For example efx/x length of box size, efy/y length of box size, efz/z length of box size. This normalizes the electric field charge to your box size. It is off when it is 0. It currently only supports *pmemd* (both the serial and MPI versions).

efphase	efphase sets the timestep phase for the electric field using the equation$\cos\left((2\pi \times effreq/1000)(dt \times step) - (\pi \times efphase/180)\right)$. It currently only supports *pmemd* (both the serial and MPI versions).
effreq	effreq sets the timestep frequency for the electric field using the equation $\cos\left((2\pi \times effreq/1000)(dt \times step) - (\pi \times efphase/180)\right)$. It currently only supports *pmemd* (both the serial and MPI versions).

17.7.2. Particle Mesh Ewald

The Particle Mesh Ewald (PME) method is always "on", unless *ntb* = 0. PME is a fast implementation of the Ewald summation method for calculating the full electrostatic energy of a unit cell (periodic box) in a macroscopic lattice of repeating images. The PME method is fast since the reciprocal space Ewald sums are B-spline interpolated on a grid and since the convolutions necessary to evaluate the sums are calculated via fast Fourier transforms. Note that the accuracy of the PME is related to the density of the charge grid (NFFT1, NFFT2, and NFFT3), the spline interpolation order (ORDER), and the direct sum tolerance (DSUM_TOL); see the descriptions below for more information.

The particle mesh Ewald (PME) method was implemented originally in Amber 3a by Tom Darden, and has been developed in subsequent versions of Amber by many people, in particular by Tom Darden, Celeste Sagui, Tom Cheatham and Mike Crowley.[365–368] Generalizations of this method to systems with polarizable dipoles and electrostatic multipoles is described in Refs. [369, 370].

The &`ewald` namelist is read immediately after the `&cntrl` namelist. We have tried hard to make the defaults for these parameters appropriate for solvated simulations. *Please take care in changing any values from their defaults.* The `&ewald` namelist has the following variables:

nfft1, nfft2, nfft3	These give the size of the charge grid (upon which the reciprocal sums are interpolated) in each dimension. Higher values lead to higher accuracy (when the DSUM_TOL is also lowered) but considerably slow the calculation. Generally it has been found that reasonable results are obtained when NFFT1, NFFT2 and NFFT3 are approximately equal to A, B and C, respectively, leading to a grid spacing (A/NFFT1, etc.) of 1.0 Å. Significant performance enhancement in the calculation of the fast Fourier transform is obtained by having each of the integer NFFT1, NFFT2 and NFFT3 values be a *product of powers* of 2, 3, and/or 5. If the values are not given, the program will chose values to meet these criteria.
order	The order of the B-spline interpolation. The higher the order, the better the accuracy (unless the charge grid is too coarse). The minimum order is 3. An order of 4 (the default) implies a cubic spline approximation which is a good standard value. Note that the cost of the PME goes as roughly the order to the third power.
verbose	Standard use is to have VERBOSE = 0. Setting VERBOSE to higher values (up to a maximum of 3) leads to voluminous output of information about the PME run.
ew_type	Standard use is to have EW_TYPE = 0 which turns on the particle mesh ewald (PME) method. When EW_TYPE = 1, instead of the approximate, interpolated PME, a *regular* Ewald calculation is run. The number of reciprocal vectors used depends upon RSUM_TOL, or can be set by the user. The exact Ewald summation is present mainly to serve as an accuracy check allowing users to determine if the PME grid spacing, order and direct sum tolerance lead to acceptable results. Although the cost of the exact Ewald method formally increases with system size at a much higher rate than the PME, it may be faster for small numbers of atoms ($<$ 500). For larger, macromolecular systems, with $>$ 500 atoms, the PME method is significantly faster.
dsum_tol	This relates to the width of the direct sum part of the Ewald sum, requiring that the value of the direct sum at the Lennard-Jones cutoff value (specified in CUT as during standard dynamics) be less than DSUM_TOL. In practice it has been found that the relative error in the Ewald forces (RMS) due to cutting off the direct sum at CUT is between 10.0 and 50.0 times DSUM_TOL. Standard values for

	DSUM_TOL are in the range of 10^{-6} to 10^{-5} , leading to estimated RMS deviation force errors of 0.00001 to 0.0005. Default is 10^{-5}.
rsum_tol	This serves as a way to generate the number of reciprocal vectors used in an Ewald sum. Typically the relative RMS reciprocal sum error is about 5-10 times RSUM_TOL. Default is 5 x 10^{-5}.
mlimit(1,2,3)	This allows the user to explicitly set the number of reciprocal vectors used in a regular Ewald run. Note that the sum goes from -MLIMIT(2) to MLIMIT(2) and -MLIMIT(3) to MLIMIT(3) with symmetry being used in first dimension. Note also the sum is truncated outside an automatically chosen sphere.
ew_coeff	Ewald coefficient, in Å^{-1} . Default is determined by *dsum_tol* and *cutoff*. If it is explicitly inputed then that value is used, and *dsum_tol* is computed from *ew_coeff* and *cutoff*.
nbflag	If *nbflag = 0*, construct the direct sum nonbonded list in the "old" way, *i.e.* update the list every *nsnb* steps. If *nbflag = 1* (the default when *imin = 0* or *ntb* $>$ *0*), *nsnb* is ignored, and the list is updated whenever any atom has moved more than 1/2 *skinnb* since the last list update.
skinnb	Width of the nonbonded "skin". The direct sum nonbonded list is extended to *cut* + *skinnb*, and the van der Waals and direct electrostatic interactions are truncated at *cut*. Default is 2.0 Å. Use of this parameter is required for energy conservation, and recommended for all PME runs.
nbtell	If *nbtell = 1*, a message is printed when any atom has moved far enough to trigger a list update. Use only for debugging or analysis. Default of 0 inhibits the message.
netfrc	The basic "smooth" PME implementation used here does not necessarily conserve momentum. If *netfrc = 1*, (the default) the total force on the system is artificially removed at every step. This parameter is set to 0 if minimization is requested, which implies that the gradient is an accurate derivative of the energy. You should only change this parameter if you really know what you are doing.
vdwmeth	Determines the method used for van der Waals interactions beyond those included in the direct sum. A value of 0 includes no correction; the default value of 1 uses a continuum model correction for energy and pressure.
eedmeth	Determines how the switch function for the direct sum Coulomb interaction is evaluated. The default value of 1 uses a cubic spline. A value of 2 implies a linear table lookup. A value of three implies use of an "exact" subroutine call. When *eedmeth=4,* no switch is used (*i.e.* the bare Coulomb potential is evaluated in the direct sum, cut off sharply at CUT). When *eedmeth=5,* there is no switch, and a distance-dependent dielectric is used (*i.e.* the distance dependence is $1/r^2$ rather than $1/r$). The last two options are intended for non-periodic calculations, where no reciprocal term is computed.
eedtbdns	Density of spline or linear lookup table, if *eedmeth* is 1 or 2. Default is 500 points per unit.
column_fft	1 or 0 flag to turn on or off, respectively, column-mode fft for parallel runs. The default mode is slab mode which is efficient for low processor counts. The column method can be faster for larger processor counts since there can be more columns than slabs and the communications pattern is less congested. This flag has no effect on non-parallel runs. Users should test the efficiency of the method in comparison to the default method before performing long calculations. Default is 0 (off).

17.7.3. Using IPS for the calculation of nonbonded interactions

Isotropic Periodic Sum (IPS) is a method for long-range interaction calculation.[371–376] Unlike the Ewald method, which uses periodic boundary images to calculate long range interactions, IPS uses isotropic periodic images of a local region to calculate the long-range contributions.

The IPS method in the current version is different from that implemented in Amber10. All IPS potentials use rationalized polynomial forms and the electrostatic interaction is calculated using the polar IPS potential.[375] In addition, the 3D IPS/DFFT algorithm [374] is implemented to handle heterogeneous systems as well as finite systems. A homogeneous system is defined as the one where a cutoff region (with *cut* as its radius) has similar composition throughout the system, such as small molecular solutions. Otherwise, a system is defined as a heterogeneous system, such as interfacial systems or finite systems. For heterogeneous systems, a local region larger than the cutoff region, normally equal or larger than the periodic boundary box, must be used to produce accurate long range interactions. For homogeneous systems, it is recommended to use the 3D IPS method (*ips*$\leq$3), which uses the cutoff distance, *cut*, to define the local region radius. *cut* is typically around 10 Å. The 3D IPS/DFFT method (*ips*$\geq$4) can be used for any type of systems, but is recommended for heterogeneous systems only due to the extra discrete fast Fourier transform (DFFT) expense.

For the amoeba polarizable potentials in *sander*, 3D IPS is implemented for interactions between charges, dipoles, and multipoles. The local region radius takes the value of *ee_dsum_cut* in the amoeba namelist, typically, 7 Å.

ips	Flag to control nonbonded interaction calculation method. The *cut* value will be used to define the local region radius for *ips*$\leq$3. When IPS is used for electrostatic interaction, PME will be turned off. When using the amoeba polarizable potentials, *iamoeba=1*, *ips*>0 (same as *ips*=2)will turn on 3D IPS for all charge, dipole, and quadrupole interactions and the *ee_dsum_cut* value will be used to define the local region radius. **= 0** IPS will not be used (default). **= 1** 3D IPS will be used for both electrostatic and L-J interactions. **= 2** 3D IPS will be used only for electrostatic, including all multipole, interactions. **= 3** 3D IPS will be used only for L-J interactions. **= 4** 3D IPS/DFFT will be used for both electrostatic and L-J interactions. **= 5** 3D IPS/DFFT will be used only for electrostatic interactions. **= 6** 3D IPS/DFFT will be used only for L-J interactions.
raips	Local region radius. *raips* is automatically set to *cut* for 3D IPS calculations (*ips*$\leq$3) and should be set larger than *cut* for 3D IPS/DFFT calculations (*ips*$\geq$4). A negative value indicates that it is set to the longest box side of a simulation system. For finite systems, i.e., system without periodic boundary conditions, *raips*=∞, which corresponding no image interaction. The default value is -1 Å.
mipsx,mipsy,mipsz	Number of grids along three periodic boundary sides when using 3D IPS/DFFT method (*ips*$\geq$4). Negative values indicate they are calculated based on the grid size, *gridips*. Typical numbers are the lengths of box sides (in Å) divided by 2 Å. Default values are -1. When *ips*=6 and PME is used for electrostatic interaction, they are set to *nfft1*, *nfft2*, and *nfft3* defined for PME, respectively.
mipso	The order of the B-spline interpolation (*ips*$\geq$4). The higher the order, the better the accuracy (unless the charge grid is too coarse). The minimum order is 3. An order of 4 (the default) implies a cubic spline approximation which is a good standard value. The cost for the DFFT calculation goes as roughly the order to the third power. For *ips*=6 and PME is used to electrostatic interaction, it is set to *order* defined for PME.
gridips	Grid size for 3D IPS/DFFT calculation (*ips*$\geq$4). The default value is 2 Å.
dvbips	Volume tolerance for updating IPS function grids (*ips*$\geq$4). When volume changes like in *NPT* simulations, the grid size changes and IPS function on grid points need be updated. The updating only happens when the volume change ratio is more than *dvbips*. The default value is 1×10^{-8}.

17.7.4. Extra point options

Several parameters deal with "extra-points" (sometimes called lone-pairs), which are force centers that are not at atomic positions. These are currently defined as atoms with "EP" in their names. These input variables are really only for the convenience of force-field developers; *do not change the defaults unless you know what you are doing, and have read the code.* These variables are set in the &ewald namelist.

frameon If *frameon* is set to 1, (default) the bonds, angles and dihedral interactions involving the lone pairs/extra points are removed except for constraints added during parm. The lone pairs are kept in ideal geometry relative to local atoms, and resulting torques are transferred to these atoms. To treat extra points as regular atoms, set frameon=0.

chngmask If *chngmask*=1 (default), new 1-1, 1-2, 1-3 and 1-4 interactions are calculated. An extra point belonging to an atom has a 1-1 interaction with it, and participates in any 1-2, 1-3 or 1-4 interaction that atom has. For example, suppose (excusing the geometry) C1,C2,C3,C4 form a dihedral and each has 1 extra point attached as below

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

The 1-4 interactions include C1-C4, Ep1-C4, C1-Ep4, and Ep1-Ep4. (To see a printout of all 1-1, 1-2, 1-3 and 1-4 interactions set verbose=1.) These interactions are masked out of nonbonds. Thus the amber mask list is rebuilt from these 1-1, 1-2, 1-3 and 1-4 pairs. A separate list of 1-4 nonbonds is then compiled. This list does not agree in general with the above 1-4, since a 1-4 could also be a 1-3 if its in a ring. See the *ephi()* routine for the precise algorithm involved here. The list of 1-4 nonbonds is printed if *verbose=1.*

17.7.5. Polarizable potentials

The following parameters are relevant for *polarizable potentials*, that is, when *ipol* is set to 1 in the &cntrl namelist. These variables are set in the &ewald namelist.

indmeth If indmeth is 0, 1, or 2 then the nonbond force is called iteratively until successive estimates of the induced dipoles agree to within DIPTOL (default 0.0001 debye) in the root mean square sense. The difference between indmeth = 0, 1, or 2 have to do with the level of extrapolation (1st, 2nd or 3rd-order) used from previous time steps for the initial guess for dipoles to begin the iterative loop. So far 2nd order (indmeth=1) seems to work best.

If indmeth = 3, use a Car-Parinello scheme wherein dipoles are assigned a fictitious mass and integrated each time step. This is much more efficient and is the current default. Note that this method is unstable for dt > 1 fs.

diptol Convergence criterion for dipoles in the iterative methods. Default is 0.0001 Debye.

maxiter For iterative methods (indmeth<3), this is the maximum number of iterations allowed per time step. Default is 20.

dipmass The fictitious mass assigned to dipoles. Default value is 0.33, which works well for 1fs time steps. If dipmass is set much below this, the dynamics are rapidly unstable. If set much above this the dynamics of the system are affected.

diptau This is used for temperature control of the dipoles (for indmeth=3). If *diptau* is greater than 10 (ps units) temperature control of dipoles is turned off. Experiments so far indicate that running the system in NVE with no temperature control on induced dipoles leads to a slow heating, barely noticeable on the 100ps time scale. For runs of length 10ps, the energy conservation with this method rivals that of SPME for standard fixed charge systems. For long runs, we recommend setting a weak temperature control (e.g. 9.99 ps) on dipoles as well as on the atoms. Note that to achieve good energy conservation with iterative method, the diptol must be below 10 -7 debye, which is much more expensive. Default is 11 ps (*i.e.* default is turned off).

irstdip If indmeth=3, a restart file for dipole positions and velocities is written along with the restart for atomic coordinates and velocities. If irstdip=1, the dipolar positions and velocities from the inpdip file are read in. If irstdip=0, an iterative method is used for step 1, after which Car-Parrinello is used.

scaldip To scale 1-4 charge-dipole and dipole-dipole interactions the same as 1-4 charge-charge (i.e. divided by scee) set scaldip=1 (default). If scaldip=0 the 1-4 charge-dipole and dipole-dipole interactions are treated the same as other dipolar interactions (i.e. divided by 1).

17.7.6. Dipole Printing

By including a &dipoles namelist containing a series of groups, at the end of the input file, the printing of permanent, induced and total dipoles is enabled.

The X, Y and Z components of the dipole (in debye) for each group will be written to mdout every NTPR steps. In order to avoid ambiguity with charged groups all of the dipoles for a given group are calculated with respect to the centre of mass of that group.

It should be noted that the permanent, inducible and total dipoles will be printed regardless of whether a *polarizable potential* is in use. However, only the permanent dipole will have any physical meaning when *non-polarizable potentials* are in use.

It should also be noted that the groups used in the dipole printing routines are not exclusive to these routines and so the dipole printing procedure can only be used when group input is *not* in use for something else (*i.e.* restraints).

17.7.7. Detailed MPI Timings

profile_mpi Adjusts whether detailed per thread timings should be written to a file called profile_mpi when running sander in parallel. By default only average timings are printed to the output file. This is done for performance reasons, especially when running *multisander* runs. However for development it is useful to know the individual timings for each mpi thread. When running in serial the value of profile_mpi is ignored.

= 0 No detailed MPI timings will be written (default).

= 1 A detailed breakdown of the timings for each MPI thread will be written to the file: profile_mpi.

17.8. Varying conditions

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.

TYPE Defines quantity being varied; valid options are listed below.

ISTEP1,ISTEP2 This change is applied over steps/iterations ISTEP1 through ISTEP2. If ISTEP2 = 0, this change will remain in effect from step ISTEP1 to the end of the run at a value of VALUE1 (VALUE2 is ignored in this case). *(default= both 0)*

VALUE1,VALUE2 Values of the change corresponding to ISTEP1 and ISTEP2, respectively. If ISTEP2=0, the change is fixed at VALUE1 for the remainder of the run, once step ISTEP1 is reached.

IINC If IINC $>$ 0, then the change is applied as a step function, with IINC steps/iterations between each change in the target VALUE (ignored if ISTEP2=0). If IINC =0, the change is done continuously. *(default=0)*

IMULT If IMULT=0, then the change will be linearly interpolated from VALUE1 to VALUE2 as the step number increases from ISTEP1 to ISTEP2. *(default)* If IMULT=1, then the change will be effected by a series of multiplicative scalings, using a single factor, R, for all scalings. i.e.

VALUE2 = (R**INCREMENTS) * VALUE1.

INCREMENTS is the number of times the target value changes, which is determined by ISTEP1, ISTEP2, and IINC.

The remainder of this section describes the options for the TYPE parameter. For a few types of cards, the meanings of the other variables differ from that described above; such differences are noted below. Valid Options for TYPE (you must use uppercase) are:

BOND	Varies the relative weighting of bond energy terms.
ANGLE	Varies the relative weighting of valence angle energy terms.
TORSION	Varies the relative weighting of torsion (and J-coupling) energy terms. Note that any restraints defined in the input to the PARM program are included in the above. Improper torsions are handled separately (IMPROP).
IMPROP	Varies the relative weighting of the "improper" torsional terms. These are not included in TORSION.
VDW	Varies the relative weighting of van der Waals energy terms. This is equivalent to changing the well depth (epsilon) by the given factor.
HB	Varies the relative weighting of hydrogen-bonding energy terms.
ELEC	Varies the relative weighting of electrostatic energy terms.
NB	Varies the relative weights of the non-bonded (VDW, HB, and ELEC) terms.
ATTRACT	Varies the relative weights of the attractive parts of the van der waals and h-bond terms.
REPULSE	Varies the relative weights of the repulsive parts of the van der waals and h-bond terms.
RSTAR	Varies the effective van der Waals radii for the van der Waals (VDW) interactions by the given factor. Note that this is done by changing the relative attractive and repulsive coefficients, so ATTRACT/REPULSE should not be used over the same step range as RSTAR.
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: $\mathrm{ISTEP1} \leq \mathrm{ABS(delta\_residue)} \leq \mathrm{ISTEP2},$

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:

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

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

TGTRMSD Varies the RMSD target value for targeted MD.

TEMP0 Varies the target temperature TEMP0.

TEMP0LES Varies the LES target temperature TEMP0LES.

TAUTP Varies the coupling parameter, TAUTP, used in temperature scaling when temperature coupling options NTT=1 is used.

CUT Varies the non-bonded cutoff distance.

NSTEP0 If present, this instruction will reset the initial value of the step counter (against which ISTEP1/ISTEP2 and NSTEP1/NSTEP2 are compared) to the value ISTEP1. This only affects the way in which NMR weight restraints are calculated. It does not affect the value of NSTEP that is printed as part of the dynamics output. An NSTEP0 instruction only has an effect at the beginning of a run. For this card (only) ISTEP2, VALUE1, VALUE2 and IINC are ignored. If this card is omitted, NSTEP0 = 0. This card can be useful for simulation restarts, where NSTEP0 is set to the final step on the previous run.

STPMLT If present, the NMR step counter will be changed in increments of STPMLT for each actual dynamics step. For this card, only VALUE1 is read. ISTEP1, ISTEP2, VALUE2, IINC, and IMULT are ignored. Default = 1.0.

DISAVE, ANGAVE, TORAVE If present, then by default time-averaged values (rather than instantaneous values) for the appropriate set of restraints will be used. DISAVE controls distance data, ANGAVE controls angle data, TORAVE controls torsion data. See below for the functional form used in generating time-averaged data.

For these cards: VALUE1 = τ (characteristic time for exponential decay) VALUE2 = POWER (power used in averaging; the nearest integer of value2 is used) Note that the range (ISTEP1→ISTEP2) applies only to TAU; The value of POWER is not changed by subsequent cards with the same ITYPE field, and time-averaging will always be turned on for the entire run if one of these cards appears.

Note also that, due to the way that the time averaged internals are calculated, changing τ at any time after the start of the run will only affect the relative weighting of steps occurring after the change in τ . Separate values for τ and POWER are used for bond, angle, and torsion averaging.

The default value of τ (if it is 0.0 here) is 1.0D+6, which results in no exponential decay weighting. Any value of $\tau \geq$ 1.D+6 will result in no exponential decay.

If DISAVE,ANGAVE, or TORAVE is chosen, one can still force use of an instantaneous value for specific restraints of the particular type (bond, angle, or torsion) by setting the IFNTYP field to "1" when the restraint is defined (IFNTYP is defined in the DISANG file).

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$\neq$0, this initial value of internal r is reset as follows:

```
-1000. < VALUE1 < 1000.: Initial value = r_initial + VALUE
 VALUE1 <= -1000.: Initial value = r_target + 1000.
 1000. <= VALUE1 : Initial value = r_target - 1000.
```

r_target is the target value of the internal, given by R2+R3 (or just R3, if R2 is 0). VALUE1 is in angstroms for bonds, in degrees for angles.

VALUE2: This field can be used to set the value of τ used in calculating the time-averaged values of the internal restraints reported at the end of a simulation (if LISTOUT is specified in the redirection section below). By default, no exponential decay weighting is used in calculating the final reported values, regardless of what value of τ was used during the simulation. If VALUE2>0, then τ = VALUE2 will be used in calculating these final reported averages. Note that the value of VALUE2 = τ specified here only affects the reported averaged values in at the end of a simulation. It does not affect the time-averaged values used during the simulation (those are changed by the VALUE1 field of DISAVE, ANGAVE and TORAVE instructions).

IINC: If IINC = 0, then forces for the class of time-averaged restraints will be calculated exactly as (dE/dr_ave) (dr_ave/dx). If IINC = 1, then then forces for the class of time-averaged restraints will be calculated as (dE/dr_ave) (dr(t)/dx). Note that this latter method results in a non-conservative force, and does not integrate to a standard form. But this latter formulation helps avoid the large forces due to the (1+*i*) term in the exact derivative calculation–and may avert instabilities in the molecular dynamics trajectory for some systems. See the discussion of time-averaged restraints following the input description. Note that the DISAVI, ANGAVI, and TORAVI instructions will have no affect unless the corresponding time average request card (DISAVE, ANGAVE or TORAVE, respectively) is also present.

DUMPFREQ Istep1 is the only parameter read, and it sets the frequency at which the coordinates in the distance or angle restraints are dumped to the file specified by the DUMPAVE command in the I/O redirection section. (For these cards, ISTEP1 and IMULT are ignored).

END END of this section.

NOTES:

1. All weights are relative to a default of 1.0 in the standard force field.
2. Weights are not cumulative.
3. For any range where the weight of a term is not modified by the above, the weight reverts to 1.0. For any range where TEMP0, SOFTR or CUTOFF is not specified, the value of the relevant constant is set to that specified in the input file.
4. If a weight is set to 0.0, it is set internally to 1.0D-7. This can be overridden by setting the weight to a negative number. In this case, a weight of exactly 0.0 will be used. *However,* if any weight is set to exactly 0.0, it cannot be changed again during this run of the program.

5. If two (or more) cards change a particular weight over the same range, the weight given on the last applicable card will be the one used.

6. Once any weight change for which NSTEP2=0 becomes active (i.e. one which will be effective for the remainder of the run), the weight of this term cannot be further modified by other instructions.

7. Changes to RSTAR result in exponential weighting changes to the attractive and repulsive terms (proportional to the scale factor**6 and **12, respectively). For this reason, scaling RSTAR to a very small value (e.g. $\leq$0.1) may result in a zeroing-out of the vdw term.

17.9. File redirection commands

Input/output redirection information can be read as described here. Redirection cards must follow the end of the weight change information. 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. cedureIf 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 23.1) will be read.
NOESY	File from which NOESY volume information (Section 23.2) will be read.
SHIFTS	File from which chemical shift information (Section 23.3) will be read.
PCSHIFT	File from which paramagnetic shift information (Section 23.3) will be read.
DIPOLE	File from which residual dipolar couplings (Section 23.5) will be read.
CSA	File from which CSA or pseduo-CSA restraints (Section 23.6) 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, then averaged values will be output. If the DUMPFREQ command has been used, the instantaneous values will be output.

17.10. Getting debugging information

The debug options in *sander* are there principally to help developers test new options or to test results between two machines or versions of code, but can also be useful to users who want to test the effect of parameters on the accuracy of their ewald or pme calculations. If the debug options are set, *sander* will exit after performing the debug tasks set by the user.

To access the debug options, include a &debugf namelist. Input parameters are:

do_debugf	Flag to perform this module. Possible values are zero or one. Default is zero. Set to one to turn on debug options.

One set of options is to test that the atomic forces agree with numerical differentiation of energy.

atomn	Array of atom numbers to test atomic forces on. Up to 25 atom numbers can be specified, separated by commas.
nranatm	number of random atoms to test atomic forces on. Atom numbers are generated via a random number generator.
ranseed	seed of random number generator used in generating atom numbers default is 71277
neglgdel	negative log of delta used in numerical differentiating; e.g. 4 means delta is 10^{-4} Angstroms. Default is 5. *Note*: In general it does no good to set nelgdel larger than about 6. This is because the relative force error is at best the square root of the numerical error in the energy, which ranges from 10^{-15} up to 10^{-12} for energies involving a large number of terms.
chkvir	Flag to test the atomic and molecular virials numerically. Default is zero. Set to one to test virials.
dumpfrc	Flag to dump energies, forces and virials, as well as components of forces (bond, angle forces etc.) to the file "forcedump.dat" This produces an ascii file. Default is zero. Set to one to dump forces.
rmsfrc	Flag to compare energies forces and virials as well as components of forces (bond, angle forces etc.) to those in the file "forcedump.dat". Default is zero. Set to one to compare forces.

Several other options are also possible to modify the calculated forces.

zerochg	Flag to zero all charges before calculating forces. Default zero. Set to one to remove charges.
zerovdw	Flag to remove all van der Waals interactions before calculating forces. Default zero. Set to one to remove van der Waals.
zerodip	Flag to remove all atomic dipoles before calculating forces. Only relevant when polarizability is invoked.
do_dir, do_rec, do_adj, do_self, do_bond, do_cbond, do_angle, do_ephi, do_xconst, do_cap	These are flags which turn on or off the subroutines they refer to. The defaults are one. Set to zero to prevent a subroutine from running. For example, set do_dir=0 to turn off the direct sum interactions (van der Waals as well as electrostatic). These options, as well as the zerochg, zerovdw, zerodip flags, can be used to fine tune a test of forces, accuracy, etc.

EXAMPLES:

This input list tests the reciprocal sum forces on atom 14 numerically, using a delta of 10^{-4}.

```
&debugf
neglgdel=4, nranatm = 0, atomn = 14,
do_debugf = 1,do_dir = 0,do_adj = 0,do_rec = 1, do_self = 0,
do_bond = 1,do_angle = 0,do_ephi = 0, zerovdw = 0, zerochg = 0,
chkvir = 0,
dumpfrc = 0,
rmsfrc = 0,
/
```

This input list causes a dump of force components to "forcedump.dat". The bond, angle and dihedral forces are not calculated, and van der Waals interactions are removed, so the total force is the Ewald electrostatic force, and the only nonzero force components calculated are electrostatic.

```
&debugf
neglgdel=4, nranatm = 0, atomn = 0,
do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
chkvir = 0,
dumpfrc = 1,
rmsfrc = 0,
/
```

In this case the same force components as above are calculated, and compared to those in "forcedump.dat". Typically this is used to get an RMS force error for the Ewald method in use. To do this, when doing the force dump use ewald or pme parameters to get high accuracy, and then normal parameters for the force compare:

```
&debugf
neglgdel=4, nranatm = 0, atomn = 0,
do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
chkvir = 0,
dumpfrc = 0,
rmsfrc = 1,
/
```

For example, if you have a 40x40x40 unit cell and want to see the error for default pme options (cubic spline, 40x40x40 grid), run 2 jobs—— (assume box params on last line of inpcrd file)

Sample input for 1st job:

```
&cntrl
dielc =1.0,
cut = 11.0, nsnb = 5, ibelly = 0,
ntx = 5, irest = 1,
ntf = 2, ntc = 2, tol = 0.0000005,
ntb = 1, ntp = 0, temp0 = 300.0, tautp = 1.0,
nstlim = 1, dt = 0.002, maxcyc = 5, imin = 0, ntmin = 2,
ntpr = 1, ntwx = 0, ntt = 0, ntr = 0,
jfastw = 0, nmrmax=0, ntave = 25,
/
&debugf
do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
chkvir = 0,
dumpfrc = 1,
rmsfrc = 0,
/
&ewald
nfft1=60,nfft2=60,nfft3=60,order=6, ew_coeff=0.35,
/
```

Sample input for 2nd job:

```
&cntrl
dielc =1.0,
cut = 8.0, nsnb = 5, ibelly = 0,
ntx = 7, irest = 1,
ntf = 2, ntc = 2, tol = 0.0000005,
ntb = 1, ntp = 0, temp0 = 300.0, tautp = 1.0,
nstlim = 1, dt = 0.002, maxcyc = 5, imin = 0, ntmin = 2,
ntpr = 1, ntwx = 0, ntt = 0, ntr = 0,
jfastw = 0, nmrmax=0, ntave = 25,
/
&debugf
do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
chkvir = 0,
dumpfrc = 0,
rmsfrc = 1,
/
&ewald
ew_coeff=0.35,
/
```

Note that an Ewald coefficient of 0.35 is close to the default error for an 8 Angstrom cutoff. However, the first job used an 11 Angstrom cutoff. The direct sum forces calculated in the 2nd job are compared to these, giving the RMS error due to an 8 Angstrom cutoff, with this value of ew_coeff. The reciprocal sum error calculated in the 2nd job is with respect to the pme reciprocal forces in the 1st job considered as "exact".

Note further that if in these two jobs you had not specified "ew_coeff" *sander* would have calculated ew_coeff according to the cutoff and the direct sum tolerance, defaulted to 10^{-5}. This would give two different ewald coefficients. Under these circumstances the direct, reciprocal and adjust energies and forces would not agree well between the two jobs. However the total energy and forces should agree reaso`nrespa`nably, (forces to within about $5x10^{-4}$ relative RMS force error) Since the totals are invariant to the coefficient.

Finally, note that if other force components are calculated, such as van der Waals, bond, angle, etc., then the total force will include these, and the relative RMS force errors will be with respect to this total force in the denominator.

17.11. multisander (and multipmemd)

The *multisander* and *multipmemd* functionality are available in the parallel versions of the programs (i.e., *sander.MPI* and *pmemd.MPI*). This mode allows multiple independent simulations, or replicas, to be run in the same program instance. It is particularly useful for computer clusters in which priority is given to large CPU-count jobs. In this case, the command-line usage of sander and pmemd is slightly altered, as shown below:

```
mpirun -np <#proc> sander.MPI -ng <#groups> -groupfile groupfile
```

In this case, `#proc` processors will be evenly divided among `#groups` individual simulations (`#proc` must be a multiple of `#group`!). The groupfile consists of a number of lines which is the command-line for each of the `#groups` simulations you wish to run. Comment lines (i.e., those with # in the first column) are ignored, after which the first `#groups` lines are read as the command-line flags of the N^{th} simulation.

The multisander and multipmemd mechanisms are also utilized for methods requiring multiple simulations to communicate with one another, such as thermodynamic integration in sander and replica exchange molecular dynamics (both described later). An example groupfile and program call are shown below.

Groupfile:

```
# Comment lines must start with a pound sign
# and there can be as many comment lines as you
# want, wherever you want them.
-O -p prmtop1 -c inpcrd1 -i replica1.mdin -suffix replica1
-O -p prmtop2 -c inpcrd2 -i replica2.mdin -suffix replica2
-O -p prmtop3 -c inpcrd3 -i replica3.mdin -suffix replica3
-O -p prmtop4 -c inpcrd4 -i replica4.mdin -suffix replica4
```

The `-suffix` flag behaves slightly differently than it does for classical use. In standard simulations (*i.e.*, without *multisander* or *multipmemd*), the provided suffix will be applied only to output files that are printed but were not given names on the command-line. With multisander, however, each thread has to produce different output files so that different replicas do not try to write to the same file. As a result, a default suffix of 000, 001, 002, etc. is given to the replicas and is added to every unspecified output file. If a `-suffix` is specified in the groupfile, as shown above, every output file—including those given an explicit name for that replica—are given the additional suffix.

The four simulations shown in the groupfile above can be run on 8 processors each with the following command (note, running *sander.MPI* may differ on your system).

```
mpirun -np 32 sander.MPI -ng 4 -groupfile groupfile
```

The *multisander* and *multipmemd* concepts are implemented via the use of MPI communicators. Each replica is assigned a replica-wide communicator along which all communications required for standard MD simulations are performed (called `commsander` and `pmemd_comm` in *sander* and *pmemd*, respectively). Each replica communicator has a master thread (rank 0 in that communicator), and the master thread of each replica are joined in another MPI communicator of replica masters (called `commmaster` and `pmemd_master_comm` in *sander* and *pmemd*, respectively). All inter-replica communication is performed via `commmaster` or `pmemd_master_comm`.

By default, all N threads are allocated to each of the M groups by dividing the threads sequentially. That is, the first N/M threads are assigned to replica 0, the second group of N/M threads are assigned to replica 1, etc. The `-ng-nonsequential` flag will stripe the thread assignments. Replica 0 will receive threads 0, $N-1$, $2N-1$, etc., while replica 1 receives threads 1, N, $2N$, etc.

17.12. APBS as an alternate PB solver in Sander

APBS is a robust, numerical Poisson-Boltzmann solver with many features (for more details see http://apbs.sourceforge.net/). APBS can be used as an alternative PB solver in sander when compiled with sander using iAPBS.[377] sander.APBS can be then used for implicit solvent MD simulations, calculation of solvation energies and electrostatic properties and to generate electrostatic potential maps for visualization. It can also be used in the MM_PBSA approach to estimate solvation and apolar (GAMMA * SASA) energy contributions to free energies of binding.

Please see APBS documentation (http://apbs.sourceforge.net/doc/user-guide/index.html) for definition of APBS input parameters and iAPBS documentation (http://mccammon.ucsd.edu/iapbs/) on how to build sander.APBS and how to use it.

To use mm_pbsa.pl script with sander.APBS the following is necessary:

- - sander.APBS must be installed in $AMBERHOME/bin directory.
- - @GENERAL and @PB sections in input file need to be modified.
- - PQR files for ligand, receptor and complex need to be prepared if an
- alternate charge/radius scheme is used (which is recommended).

Input file description

The mm_pbsa.in input file which is included in the Amber distribution can be used with the following modifications:

(1) Turn on PB and turn off GB and MS calculations in the @GENERAL section of the input file:

```
@GENERAL
MM 1
GB 0
PB 1
MS 0
```

(2) Input file @PB section:

```
#
@PB
#
#
# PROC = 3 uses sander.APBS as the PB solver
# REFE - REFE = 0 is always used with sander.APBS
# INDI and EXDI are solute and solvent dielectric constants
# SCALE - grid spacing in number of grid points per A
# LINIT - no effect
# PRBRAD - solvent probe radius in A
# ISTRNG - ionic strength in mM
#
# RADIOPT - option to set up radii and charges for PB calculation:
# 0: uses the radii from prmtop files
# 2: reads in PQR files with radii/charges information from
# lig.pqr, rec.pqr and com.pqr PQR files
#
# APBS options:
# BCFL, SRFM, CHGM, SWIN, GAMMA - see APBS and iAPBS documentation for details
# GAMMA is surface tension for apolar energies (in kJ/mol/A**2),
# defaults to 0.105 (Please note the units!)
#
PROC 3
REFE 0
INDI 1.0
EXDI 80.0
SCALE 2
LINIT 1000
PRBRAD 1.4
ISTRNG 0.0
#
RADIOPT 0
#
BCFL 2
SRFM 1
CHGM 1
SWIN 0.3
GAMMA 0.105
#
```

PQR files

With RADIOPT=2 three PQR files are required: lig.pqr, rec.pqr and com.pqr with charge/radius information for the ligand, receptor and complex, respectively. This is the recommended option to get better estimates of solvation energies.

The PQR files can be created with pdb2pqr utility:

```
pdb2pqr.py --assign-only --ff=amber com.pdb com.pqr
pdb2pqr.py --assign-only --ff=amber rec.pdb rec.pqr
pdb2pqr.py --assign-only --ff=amber lig.pdb lig.pqr
```

where –ff=amber is the requested force field charge/radius parameters. Several options are available (Amber, CHARMM, PARSE, etc.) and also a user defined charge/radius scheme is supported (with –ff=myff option).

pdb2pqr.py can be obtained from http://pdb2pqr.sourceforge.net/. PDB2PQR service is also available on the web at http://nbcr.net/pdb2pqr/. The PDB files (com.pdb, rec.pdb and lig.pdb) can be generated using ambpdb utility.

17.13. Programmer's Corner: The sander API

By Jason M. Swails

This section describes a new feature of *sander*—an application programmer interface (API) that encapsulates some of *sander*'s basic functionality into a library that can be included in your own programs. *sander* was originally written in Fortran as a standalone program that made extensive use of common blocks (i.e., global variables) and uses MPI for the parallel implementation rather than a type of shared-memory parallelization scheme like OpenMP or pthreads. This design conferred a number of constraints on the resulting API.

1. Only one system can be set up for use with the API at a time. Switching Hamiltonians or input parameters requires a lot of deallocation and reallocation and will be inefficient if done very frequently.
2. Only serial execution is supported.
3. LES and non-LES functionality cannot be combined in the same library.
4. File names have a fixed maximum length (256 characters). This can be extended only by adjusting the *sander* source code and recompiling.

Despite these limitations, the *sander* API provides functionality unavailable in other libraries, including QM/MM forces and energies, PB and GB energies, and LES functionality (through a separate library). Although originally written in Fortran, an API has been provided for four languages: Fortran, C, C++, and Python.

The next sections describe the general API design and then the Fortran, C, C++, and Python APIs specifically.

17.13.1. General API Design

This section describes the functions that are available in each variant of the *sander* API. The exact syntax for how the various functions and subroutines are called—and what, if anything, they return—is listed in the following sections for each API.

17.13.1.1. Data Structures

The following data structures are provided as part of the API. These data structures provide a way to provide input or query output from the API. They are the equivalent of C `struct`s (Fortran `type` and Python `class`). All floating point data types are double precision (`double` in C and C++, `double precision` in Fortran, and `float` in Python). All integer data types are standard integers (`int` in C and C++, `integer` in Fortran, and `int` in Python).

sander_input

This contains variables used to provide input for the sander API. The attributes that are exposed here have the same name, options, and function as the input options with the same name described earlier in this chapter (and some in earlier chapters). These attributes are listed below, with their data type (float or integer) listed in parentheses at the end.

extdiel External dielectric constant for GB calculations. (float)

intdiel Internal dielectric constant for GB calculations. (float)

rgbmax Distance cutoff in Angstroms to use when computing effective GB radii. (float)

saltcon Salt concentration, in Molarity, to use when modeling ionic strength effects in a GB calculation. (float)

cut Nonbonded cutoff in Angstroms. (float)

dielc Dielectric constant to use for all electrostatic interactions. You should use extdiel or intdiel, described above, for GB calculations (this option should only be used if you are sure it is what you want—it is usually not what you want). (float)

rdt This is an option specific to GB calculations with LES (and only has an effect when using the sanderles library). When using GB with LES, non-LES atoms require multiple effective radii due to alternate descreening effects from the different copies. When the multiple radii differ by less than rdt, only a single radius will be used for this atom. Default is 0.0. See Chapter 24 for more information. (float)

igb GB model to use for GB calculations. Allowable values are 0 (no GB), 1, 2, 5, 6 (vacuum), 7, 8, and 10 (PB). More information is available on page 59. (integer)

alpb If 1, use the analytical linearized Poisson-Boltzmann approximation. See Section 4.2 for more information. (integer)

gbsa If set to 0, no SASA-based nonpolar free energy of solvation correction is used. If set to 1, the SASA is approximated using the linear combination of pairwise overlaps method (LCPO). If set to 2, the SASA is approximated using a recursive algorithm constructing spheres around each atom. Note, gradients (forces) are not available from this model, so the forces returned by the API will be incorrect if this option is used. (integer)

lj1264 If 1, use the 12-6-4 Lennard Jones potential form designed for divalent metal ions. If 0, do not use the 12-6-4 Lennard-Jones model. The topology file must be set up correctly to use the 12-6-4 model first! (integer)

ipb Option to compute the solvation free energy using the Poisson-Boltzmann equation. Allowable values are 0 (no PB equation), 1, 2, and 4. See Chapter 6 for more information. (integer)

vdwmeth For periodic simulations only (i.e., when *ntb*, below, is set to 1). When set to 1, a long-range dispersion correction based on an analytical integral assuming an isotropic, uniform bulk particle distribution beyond the cutoff is added to the van der Waals energy. When set to 0, no correction is used. (integer)

ew_type For periodic simulations only (i.e., when *ntb*, below, is set to 1). When set to 0, the particle-mesh Ewald method is used to compute long-range electrostatics. When set to 1, a traditional Ewald method is used to compute long-range electrostatics (PME is *much* faster for systems with more than 500 atoms or so). (integer)

ntb If set to 0, periodic boundaries are not applied. If set to 1, periodic boundaries are used. The value of 2 does not (yet) apply to the API (i.e., constant pressure), as this is an MD-specific option. (integer)

ifqnt If set to 1, a QM/MM potential is used (and you must provide a set of valid QM options as well). If set to 0, no QM/MM potential is used. (integer)

jfastw Fast water definition flag. By default, the system is searched for water residues, and special routines are used to SHAKE these systems (i.e., they are constrained using the analytical SETTLE algorithm). If set to 0, this default behavior is triggered. If set to 4, the numerical SHAKE routines are used. (integer)

ntf Flag to determine which, if any, interactions to omit from the energy calculation. (integer)

ntc Flag to determine whether to use the SHAKE algorithm to constrain bond distances. (integer)

There are two subroutines that will initialize a sander_input instance with default values—one that prepares the input for a periodic simulation and one that prepares the input for an aperiodic system (either gas phase or GB calculation). The default values assigned are summarized in table 17.1.

Table 17.1.: *Summary of default values assigned to `sander_input` variables by the two initialization subroutines provided in the* sander *API. When alternate values are given for `gas_sander_input`, the latter corresponds to the value assigned if a GB model is requested.*

`sander_input` variable	`gas_sander_input` default	`pme_sander_input` default
extdiel	1 or 78.5	0
intdiel	1	0
rgbmax	25.0	25.0
saltcon	0	0
cut	1000.0	8.0
dielc	1	1
rdt	0	0
igb	6 or input value	0
alpb	0	0
gbsa	0	0
lj1264	0	0
ipb	0	0
inp	0	0
vdwmeth	0	1
ew_type	0	0
ntb	0	1
ifqnt	0	0
jfastw	0	0
ntc	1	1
ntf	1	1

qmmm_input_options

This struct contains a set of options for controlling what portion of the system is treated using quantum mechanics (QM), which QM Hamiltonian is used to treat the QM portion of the system, how the boundary between the QM and MM portions of the system are handled, and how the QM and MM portions interact.

The variables in this data structure have the same name and function as the variables defined in the `&qmmm` namelist of the input file. You can find more information about QM/MM in Chatper 10 on page 146 and about the options specifically in Chapter 9 and Subsection 10.1.6.

There are three types of data types in this struct. Floating point, integer, and character array (i.e., string) values. Like with sander_input above, floating point numbers are represented in full double precision and integers as standard integers. The strings in this section are fixed-size arrays of characters. The type of each variable is indicated in parentheses after the variable is defined followed by the fixed-size length of the array if it is an array value. The standard value for the maximum number of QM atoms (`MAX_QUANTUM_ATOMS`) is 10,000.

Note that strings are treated by the API as Fortran strings, *not* C-style strings. The main difference is that Fortran strings do not have a null terminal character (`'\0'`), which means that every character after the final "letter" of the string must contain a space (or null character). As a result, the typical string routines defined in the C `string.h` header file (e.g., `strcpy` and `strncpy`) may not assign the strings correctly if they are not properly initialized entirely with spaces first. That is why the `qm_sander_input` function is provided as part of the API, so I suggest that you always initialize a `qmmm_input_options` data structure using this method when using the C or C++ APIs.

The defaults listed below are those assigned by the `qm_sander_input` function in the API (and are the same as the defaults defined in Subsection 10.1.6).

qmcut Nonbonded cutoff in Angstroms used for QM/MM nonbonded interactions (note there is no such thing as a cutoff *within* the QM region, since it is the wavefunction of the entire system we are optimizing). The default value is the MM cutoff being used (i.e., `cut` from `sander_input`, above). (float)

lnk_dis Distance in Angstroms of the QM atom to its link atom. Default is 1.09. (float)

scfconv Controls the convergence of the SCF calculation. The SCF terminates when the energy difference between the last two steps is smaller than the value given here. Default is 10^{-8} and the smallest value that can practically be used within the limits of double precision floating point arithmetic is 10^{-14}. (float)

errconv SCF tolerance on the maximum absolute value of the erro matrix (i.e., the commutator of the Fock matrix with the density matrix). The value is in units of Hartrees. The default value is large enough that scfconv will always be more strict. (float)

dftb_telec Electronic temperature, in K, used to accelerate SCC convergence in DFTB calculations. The electronic temperature affects the Fermi distribution promoting some HOMO/LUMO mixing, which can accelerate the convergence in difficult cases. In most cases, a low *telec* (around 100K) is enough. Should be used only when necessary, and the results checked carefully. Default: 0.0 (float)

dftb_telec_step The size of the step to take when reducing the electronic temperature in a DFTB calculation. The smaller the step, the longer it will take to get the electronic temperature to zero. (float)

fockp_d1 First prefactor for the Fock matrix prediction. Default is 2.4. Changing this is not recommended. (float)

fockp_d2 Second prefactor for the Fock matrix prediction. Default is -1.2. Changing this is not recommended. (float)

fockp_d3 Third prefactor for the Fock matrix prediction. Default is -0.8. Changing this is not recommended. (float)

fockp_d4 Fourth prefactor for the Fock matrix prediction. Default is 0.6. Changing this is not recommended. (float)

damp SCF damping factor. Default is 1.0. Changing this is not recommended. (float)

vshift Controls level shifting for NDDO methods (not DFTB). Virtual orbitals can be shifted up by vshift (in eV) to improve SCF convergence in cases with a small HOMO/LUMO gap. Default is 0.0. (float)

kappa Related to the Debye salt concentration for GB models. This is set automatically from saltcon in the `sander_input` data structure. (float)

pseudo_diag_criteria Controls whether a pseudo-diagonalization of the Fock matrix can be performed (not applicable for DFTB). Default is 0.05. (float)

min_heavy_mass The smallest value, in atomic mass units, that an atomic mass can have and still be considered a “heavy-atom” (i.e., anything besides Hydrogen). Default is 4.0. (float)

r_switch_hi If qmmm_switch (below) is turned on, this is the distance, in Angstroms, at which the switch goes to zero. By default, it is the same as qmcut. (float)

r_switch_lo If qmmm_switch (below) is turned on, this is the distance, in Angstroms, at which the switch turns on. By default, it is 2 Angstroms smaller than `r_switch_hi`. (float)

iqmatoms List of atom indexes, starting from 1, that will be treated using QM. This is one way, along with qmmask, of specifying the QM region. Default is an empty list. (integer array, `MAX_QUANTUM_ATOMS`).

qmgb Specifies how the QM region should be treated with Generalized Born. (integer)

- **= 2** (default) As described above, the electrostatic and “polarization” fields from the MM charges and the exterior dielectric, respectively, are included in the Fock matrix for the QM Hamiltonian.
- **= 3** This is intended for debugging and is only useful for single-point calculations. This computes the GB energy by treating every atom in the QM region as a point charge equal to its Mulliken charge. This can be compared to the result when qmgb is set to 2 to evaluate the “strain” energy from the GB solvation.

lnk_atomic_no The atomic number of the element you wish to use as the link atom. Default is 1 (Hydrogen). (integer)

ndiis_matrices The number of error vectors to use for the DIIS convergence algorithm. Default is 6. (integer)

ndiis_attempts The number of iterations that DIIS extrapolation will be attempted. Not available for DFTB. Default value is 0, maximum is 1000. (integer)

lnk_method The method used to define how classical valence terms across the QM/MM boundary will be treated. See Subsection 10.1.7 for more information. Default is 1. (integer)

qmcharge The net charge of the QM region. Default is 0. (integer)

corecharge The net charge of the core QM region. Default is 0. (integer)

buffercharge The net charge of the buffer QM region. Default is 0. (integer)

spin Spin multiplicity of the QM region. Default is 1 (singlet). (integer)

qmqmdx Controls whether QM-QM derivatives are computed analytically or pseudo-numerically. The default (and recommended) is to use analytical QM-QM derivatives. Set to 1 for analytical derivatives, 2 for pseudo-numerical derivatives. Default is 1. (integer)

verbosity This has no effect on the API, since output is suppressed. Keep the default value of 0. (integer)

printcharges This has no effect on the API since output is suppressed. Keep the default value of 0. (integer)

printdipole This has no effect on the API, since output is suppressed. Keep the default value of 0. (integer)

print_eigenvalues This has no effect on the API, since output is suppressed. Keep the default value of 0. (integer)

peptide_corr If set to 0, (default), do not apply a correction to peptide linkages. If set to 1, apply a MM correction to peptide linkforages. (integer)

itrmax Maximum number of SCF iterations to perform before deciding that the convergence has failed. Default is 1000. (integer)

printbondorders This has no effect on the API, since output is suppressed. Keep the default value of 0. (integer)

qmshake Controls whether SHAKE is applied to QM atoms. If 0, no SHAKE. If 1 (default), SHAKE QM atoms if MM SHAKE is turned on. By default, MM SHAKE is not turned on. This really has no effect, anyway, since the API does not currently support dynamics. (integer)

qmmmrij_incore If set to 1 (default), store QM-MM pairs and related equations in memory. If set to 0, do not. (integer)

qmqm_erep_incore If set to 1 (default), store QM-QM 1-electron repulsion integrals to memory. If set to 0, calculate them on-the-fly. (integer)

pseudo_diag If set to 1 (default), allow the use of pseudo-diagonalization of the Fock matrix as long as the `pseudo_diag_criteria` is met. (integer)

qm_ewald Specifies how the long-range electrostatics for the QM region should be treated. See the description in Subsection 10.1.6 for more information. (integer)

qm_pme If 0, use a regular Ewald sum for computing QM-QM and QM-MM long-range electrostatic interactions. If 1 (default), use PME instead. (integer)

kmaxqx Number of K-space vectors to use in the Ewald/PME calculations in the X-dimension. Default value is 8. (integer)

kmaxqy Same as above, but in the Y-dimension. (integer)

kmaxqz Same as above, but in the Z-dimension. (integer)

ksqmaxsq Specifies the maximum number of K^2 values for the spherical cutoff in reciprocal space when doing a QM-MM Ewald sum. The default value of 100 should be optimal for most systems. (integer)

qmmm_int Controls the way in which the QM-MM interaction is handled. See Subsection 10.1.6 for more information. Default is 1. (integer)

adjust_q Controls how charge is conserved during a QM/MM calculation with respect to link atoms. See Subsection 10.1.6 for more information. Default is 2. (integer)

tight_p_conv Controls the tightness of the convergence criteria on the density matrix in the SCF. If 0 (default), the convergence is loose. If set to 1, convergence is tight. See Chapter 9 for more information. (integer)

diag_routine The diagonalization routine to use to diagonalize the Fock matrix. By default (diag_routine = 0), the fastest routine is chosen. See the description in Chapter 9 for more details. (integer)

density_predict If 1, use the density matrix from the previous MD step. Since MD is not currently supported in the API, do not deviate from the default value of 0. (integer)

fock_predict If set to 0, do not attempt to predict the Fock matrix. (Default). If set to 1, try to. (integer)

vsolv If set to 1, use variable solvent QM/MM. If set to 0 (default), do not. This option is irrelevant to the API since it does not support QM/MM. (integer)

dftb_maxiter The maximum number of SCF iterations to be used in SCC-DFTB calculations. Default is 70. (integer)

dftb_disper If set to 1, use a dispersion correction for DFTB/SCC-DFTB. If set to 0 (default), do not. (integer)

dftb_chg Has no effect on the API, since printing is disabled. (integer)

abfqmmm Toggles the adaptive biased force QM/MM. Since the API does not support MD, this option has no effect. Default is 0. (integer)

hot_spot If set to 1, activates hot spot-like adaptive calculation in which the forces of atoms in the buffer region are linear combinations of the forces obtained from the extended and reduced calculations using a smoothing function. If set to 0 (default), disable this behavior. (integer)

qmmm_switch If set to 1, use a switching function defined by `r_switch_lo` and `r_switch_hi`. If set to 0 (default), do not. (integer)

core_iqmatoms A list of atom indices (starting at 1) that are selected for inclusion in the core QM/MM region in adaptive simulations. (integer array, `MAX_QUANTUM_ATOMS`)

buffer_iqmatoms A list of atom indices (starting at 1) that are selected for inclusion in the buffer QM/MM region in adaptive simulations. (integer array, `MAX_QUANTUM_ATOMS`)

qmmask An Amber selection mask that provides another way of defining the QM region instead of `iqmatoms`. (character array, 8192)

coremask An Amber selection mask that provides another way of defining the core QM region in adaptive simulations instead of `core_iqmatoms`. (character array, 8192)

buffermask An Amber selection mask that provides another way of defining the buffer QM region in adaptive simulations instead of `buffer_iqmatoms`. (character array, 8192)

centermask An Amber selection mask that defines the center region. If not set, it defaults to `coremask`. (character array, 8192)

dftb_3rd_order Specifies the 3rd-order DFTB correction. Default ('NONE') means no 3rd order correction is used. See Chapter 9 for more information. (character array, 256)

qm_theory String that defines which level of QM theory to use. There is no default and this must be supplied. Available options are defined in Chapter 9. (character array, 12)

pot_ene

This data structure is populated when the energy and forces are computed for the positions that are currently set. All elements of this data structure are double-precision floating point numbers and are given in kilocalories per mole.

tot The total potential energy

vdw The van der Waals contribution to the total energy (not including 1-4 interactions)

elec The electrostatic contribution to the total energy (not including 1-4 interactions)

gb Polar solvation free energy from GB calculations

bond The energy contribution from valence bonds.

angle The energy contribution from valence angles.

dihedral The energy contribution from valence torsions.

vdw_14 The energy contribution from 1-4 van der Waals interactions

elec_14 The energy contribution from 1-4 electrostatic interactions

constraint Really misnamed, this is the total restraint energy if NMR or positional restraints are used.

polar Polarization energy if you are using a polarizable force field.

hbond The 10-12 contribution to the total energy (not used in modern force fields)

surf The non-polar solvation free energy contribution from GB and PB calculations.

scf The QM energy contribution (includes charge-charge interactions between MM and QM atoms, but not dispersion interactions—those are added to the vdw component).

disp Dispersion energy contribution (?? not really sure what this is)

dvdl Not really applicable to the API, since it is used for constant pH MD calculations. This should always be 0.

angle_ub For CHARMM force field, this is the Urey-Bradley contribution to the total energy.

imp For CHARMM force field, this is the improper torsion contribution to the total energy.

cmap For CHARMM force field, this is the correction map energy contribution for coupled torsions.

emap When fitting to an electron density map, this is the restraint energy derived from violations to the map.

les The total energy contributed by the LES copies.

noe The energy penalty for NOE violations.

pb The total polar solvation free energy from PB calculations.

rism The total solvation free energy from 3D-RISM calculations.

ct Charge transfer energy (for crg_reloc)

amd_boost This is the AMD boosting energy. It is not applicable for the API since molecular dynamics is not currently supported.

17.13.1.2. Basic subroutines

This section describes the functions and subroutines that are defined by the API and explains what they do. Since their exact behavior (e.g., their arguments and return values) differ depending on which API you are using, the exact usage is deferred to later sections. However, what they *do* is described here.

There are very strong similarities between the C/C++ and Fortran function calls. While the Python function calls are also similar, the Python behavior often differs the most.

gas_sander_input This function will initialize a `sander_input` data structure with the appropriate defaults for carrying out either a gas-phase calculation or an implicit solvent GB calculation. It takes an integer argument defining the GB model to use. See the `igb` variable in the `sander_input` data structure above for allowable values.

It is recommended that you initialize your `sander_input` instance using either this routine or `pme_sander_input` to make sure that all variables are initialized. Uninitialized variables in any of the compiled languages (i.e., not Python) take on undefined behavior and could result in strange bugs.

This can be called regardless of whether or not a system is currently set up.

pme_sander_input This function will initialize a `sander_input` data structure with the appropriate defaults for carrying out a PME calculation on a periodic system. It is recommended that you initialize your `sander_input` instance using either this routine or `gas_sander_input` to make sure that all variables are initialized. Uninitialized variables in any of the compiled languages (i.e., not Python) take on undefined behavior and could result in strange bugs.

This can be called regardless of whether or not a system is currently set up.

qm_sander_input This function will initialize a `qmmm_input_options` data structure with the defaults listed in Subsection 17.13.1.1. This is the recommended method for initializing QM input options, particularly in the C and C++ interfaces where string handling is fragile.

sander_setup These functions take a topology file, coordinates, box dimensions, and a set of input options (`sander_input` and `qmmm_input_options`) and sets up the *sander* API so that energies and forces can be calculated.

These functions can only be called if no system is currently set up. You must call `sander_cleanup` before setting up a different system (or changing input parameters).

set_positions This function takes an array of double precision particle positions (3 × natom) and sets them as the active conformation.

This function can only be called if there is currently a system set up.

set_box This function takes three box lengths and the angles between them and sets the unit cell (and reciprocal unit cell) vectors from these values.

This function can only be called if there is currently a system set up.

sander_natom This function returns the number of atoms present in the system that is currently set up.

This function can only be called if there is currently a system set up.

get_positions This function returns the currently active atomic coordinates for the system that is currently set up.

This function can only be called if there is currently a system set up.

get_inpcrd_natom This function takes the name of an inpcrd file and reads the number of atoms that are defined in this file. If this file is not present or its format cannot be determined, the number of atoms is set to -1, which indicates an error.

This function can be called regardless of whether or not a system is currently set up.

read_inpcrd_file This function takes the name of an inpcrd file, an array of length 3 × natom double precision floating point numbers, and an array of 6 double precision floating numbers and fills them with the atomic coordinates and box dimensions, respectively. The box dimensions are stored as a, b, c, α, β, and γ.

Since the two arrays must already be allocated, the typical workflow is to call `get_inpcrd_natom` to determine how large the coordinate array must be made. Then call `read_incprd_file` after allocating the coordinate array.

This function can be called regardless of whether or not a system is currently set up.

is_setup This function returns whether a system is currently set up or not.

This function can be called regardless of whether or not a system is currently set up.

energy_forces This function computes the energy and forces for the current coordinates of the system that is currently set up and returns them in the potential energy data structure and (3 × natom)-length double precision array that is passed to this routine.

This function can only be called if a system is currently set up.

sander_cleanup This function clears all of the internal memory initialized and allocated by the sander_setup routines. This function can only be called if a system is set up, but after this function completes, a system is no longer set up.

17.13.2. The Fortran API

The Fortran API is implemented with a Fortran module. The module is compiled when AmberTools is built and the modulefile is deposited in `$AMBERHOME/include`.

One of the limitations of Fortran modules is that you *must* use the same compiler to build your program as you used to compile AmberTools in the first place. If you wish to change compilers (in many cases, this also includes compiler versions as well), then you need to recompile AmberTools with that same compiler as well. The available API modules are `sander_api` for the standard *sander* functionality and `sanderles_api` for the LES capabilities.

17.13.2.1. Data structures

The Fortran data structures are all different sequence `type`s. The sequence descriptor simply means that they are layed out sequentially in memory in exactly the same way that a struct is in C or C++. Variables within a type are accessed using the `%` operator.

The sander input options are available as `type(sander_input)`, the QM/MM input options are available as `type(qmmm_input_options)`, and the potential energy data structure is available as `type(potential_energy_rec)`. The names of the variables that make up each of these types are the same as those defined in Subsection 17.13.1.1.

An example of using the `sander_input` type is shown in the small code fragment below

```
use sander_api, only : sander_input
type(sander_input) :: inp
inp%cut = 9999.d0
inp%ifqnt = 0
inp%igb = 5
```

17.13.2.2. Function call syntax

This section details the function calls for the various subroutines available in the Fortran API. All of these subroutines and functions are public members of both `sanderapi_mod` and `sanderlesapi_mod`.

```
subroutine gas_sander_input(sander_input inp, int igb)
```

This subroutine takes a `sander_input` instance and optionally an integer corresponding to the GB model you want to use (see the description for igb above with regards to permissible values). If an illegal `igb` value is provided, a warning is printed to `stderr` and a value of 6 (corresponding to vacuum) is given to `inp%igb`. See table 17.1 for a list of the default values assigned to each variable.

```
subroutine pme_sander_input(sander_input inp)
```

This subroutine takes a `sander_input` instance and initializes every variable inside with the value listed in table 17.1.

```
subroutine qm_sander_input(qmmm_input_options inp)
```

This subroutine takes a `qmmm_input_options` instance and initializes all of the variables to the values given in Subsection 17.13.1.1. This is the recommended way to initialize the QM/MM options type.

```
subroutine sander_setup(character(len=*) top,
                        double precision, dimension(3*natom) crd,
                        double precision, dimension(6) box,
                        sander_input inp,
                        qmmm_input_options qm_inp,
                        integer ierr)
```

This subroutine sets up *sander* with the given topology file name, given coordinates, given box dimensions, input options, and QM/MM input options. The ierr variable is an error flag and will come back with a value of 0 if the setup succeeded or a value of 1 if it failed. Every other variable is input and guaranteed not to change.

No checking is done to make sure that the number of coordinates provided is correct compared to the number of atoms defined in the topology file. Note that answers will be ridiculous if the coordinate order does not match the atom order in the topology file. Segfaults and other memory violations are possible if the provided coordinate array or box array are too small.

The box array is given in the format $(a,b,c,\alpha,\beta,\gamma)$, which is the same as the format used at the bottom of the input coordinate and restart files. This argument is required even if the system is not periodic, but the values are not used (so they can be initialized to anything).

The `qmmm_input_options` variable is optional, but must be present if `inp%ifqnt` is 1. If `qm_inp` is not provided but QM/MM is requested, an error message will be printed to stderr and `ierr` will return with a value of 1.

The error flag `ierr` is required. If `qm_inp` is omitted, then ierr must be specified via keyword. See the examples at the end of this section. This function should never be called if a system is already set up.

```
subroutine set_positions(double precision, dimension(3*natom) crd)
```

This subroutine sets the current positions of the active system (and so can only be called if a system is currently set up). Note that the onus is on the programmer to make sure that the coordinate array is large enough. No error checking is done. The input parameter is guaranteed not to change.

```
subroutine set_box(double precision a, double precision b,
                   double precision c,
                   double precision alpha, double precision beta,
                   double precision gamma)
```

This subroutine sets the box dimensions and angles from the input parameters (which are guaranteed not to change).

```
subroutine get_positions(double precision, dimension(3*natom) positions)
```

This subroutine stores the currently active positions inside the passed array.

```
subroutine energy_forces(type(potential_energy_rec) ener,
                         double precision, dimension(3*natom) forces)
```

This subroutine will compute the energies and forces from the current conformation of the system that is currently set up and populate the `ener` type and `forces` array with the resulting values. Those parameters are purely output. The energies are all given in units of kilocalories per mole and forces are given in $kcal/mol/Å$. This subroutine can only be called if a system is currently set up.

```
subroutine sander_cleanup()
```

This subroutine will deallocate all memory used by the *sander* API and return it to a state where no system is set up and a new one can be initialized.

```
logical function is_setup()
```

This function can be called to query whether there is currently a system set up for the *sander* API. It returns `.true.` if a system is set up and `.false.` otherwise.

```
subroutine sander_natom(integer natom)
```

This subroutine will query the currently set up system and return the number of atoms defined by the topology file used during setup. The input parameter will return with the number of atoms in the system or 0 if no system is currently set up.

```
subroutine get_inpcrd_natom(character(len=*) filename, integer natom)
```

This subroutine will open the specified file and try to read how many atoms are defined in that coordinate file. It supports both NetCDF and standard ASCII-formatted inpcrd and restart files. If there was an error in reading the file—either because the file does not exist, read permissions are not set, or the format is unrecognized—`natom` will return with a value of -1. Otherwise, `natom` returns with the number of atoms defined in the inpcrd file, `filename`.

```
subroutine read_inpcrd_file(character(len=*) filename,
                            double precision, dimension(3*natom) crd,
                            double precision, dimension(6) box,
                            integer ierr)
```

This subroutine will read the specified coordinate file and fill the `crd` and `box` arrays with the coordinates and box defined in the file. Both NetCDF restart files and ASCII restart files are supported. If no box is defined in the specified file, the `box` array is initialized to 0. The coordinate array is expected to be allocated with the appropriate amount of space. You can call `get_inpcrd_natom` (described above) to determine how large the coordinate array must be.

If there is a problem reading the file—either because the file does not exist, read permissions are not set, or the format is unrecognized—`ierr` will come back with a value of 1 and the `crd` array will be uninitialized (the `box` array will still be set to 0). If reading succeeded, `ierr` will come back as 0. This function can be called regardless of whether a system is currently set up or not.

17.13.2.3. Example uses of the Fortran API

In this section we show a series of example programs that use the *sander* Fortran API. At the end of this section, we show how to compile your program using the same Fortran compiler you used to build Amber. We will assume that you created a file with the same name as the program name using the `.F90` suffix. You are recommended to use this suffix for your own programs.

We do not do any error checking in these programs since it adds considerably to the length of the example programs. However, you are encouraged to make use of the error reporting in your own programs to avoid program crashes. Syntax highlighting is applied to make the code easier to read.

The first example we provide below shows a sample program that computes purely MM energies for a non-periodic system using one of the GB models available in Amber.

```
program sample1
   use sander_api, only: sander_input, gas_sander_input, &.
                         sander_setup, energy_forces, &
                         sander_cleanup, potential_energy_rec, &
                         get_inpcrd_natom, read_inpcrd_file, &
                         sander_natom
   implicit none
   double precision, allocatable, dimension(:) :: crd, frc
   double precision, dimension(6) :: box
   type(sander_input) :: inp
   type(potential_energy_rec) :: ene
   integer :: natom, ierr
   ! Find how many atoms are in our inpcrd file
   call get_inpcrd_natom("inpcrd", natom)
   allocate(crd(natom*3), stat=ierr)
   ! Parse the inpcrd file
   call read_inpcrd_file("inpcrd", crd, box, ierr)
   ! Set up input options to use igb=5 with 0.2M salt
   call gas_sander_input(inp, 5)
   inp%saltcon = 2.0d-1
   ! Set up our system
   call sander_setup("prmtop", crd, box, inp, ierr=ierr)
   ! Coordinate array is no longer needed
   deallocate(crd)
   ! Find out how big our force array must be
   call sander_natom(natom)
   allocate(frc(natom*3), stat=ierr)
   call energy_forces(ene, frc)
   ! Do whatever you want with the energies and forces
   ! ...
   ! Free up our memory
   call sander_cleanup
   deallocate(frc)
   return
end program sample1
```

The second example we provide shows how to use the Fortran API to compute the energy for a periodic system using a multiscale QM/MM Hamiltonian. We will treat residues 10, 11, 12, and 20 using the PDDG-PM3 Hamiltonian.

```
program sample2
   use sander_api, only: sander_input, pme_sander_input, &
                         sander_setup, energy_forces, &
                         sander_cleanup, potential_energy_rec, &
                         get_inpcrd_natom, read_inpcrd_file, &
                         sander_natom, qmmm_input_options, &
                         qm_sande.r_input
   implicit none
   double precision, allocatable, dimension(:) :: crd, frc
   double precision, dimension(6) :: box
   type(sander_input) :: inp
   type(qmmm_input_options) :: qm_inp
   type(potential_energy_rec) :: ene
   integer :: natom, ierr
```

```
    ! Find how many atoms are in our inpcrd file
    call get_inpcrd_natom("inpcrd", natom)
    allocate(crd(natom*3), stat=ierr)
    ! Parse the inpcrd file
    call read_inpcrd_file("inpcrd", crd, box, ierr)
    ! Set up input options to use PME with a 10A cutoff
    call pme_sander_input(inp)
    inp%cut = 10.d0
    inp%ifqnt = 1
    call qm_sander_input(qm_inp)
    qm_inp%qmmask = ":10-12,20"
    qm_inp%qm_theory = "PDDG-PM3"
    ! Set up our system
    call sander_setup("prmtop", crd, box, inp, qm_inp, ierr)
    ! Coordinate array is no longer needed
    deallocate(crd)
    ! Find out how big our force array must be
    call sander_natom(natom)
    allocate(frc(natom*3), stat=ierr)
    call energy_forces(ene, frc)
    ! Do whatever you want with the energies and forces
    ! ...
    ! Free up our memory
    call sander_cleanup
    deallocate(frc)
    return
end program sample2
```

To compile Fortran programs using the *sander* API, the compiler must be able to find the `sander_api` (or `sanderles_api`) module files, which are deposited in `$AMBERHOME/include` when you build AmberTools. You must also link `libsander.so` (or `libsanderles.so`) when you link your program. On Mac OS X, these shared libraries are named `libsander.dylib` and `libsanderles.dylib` instead.

The programs we've written above are simple enough that they can be compiled and linked at the same time. The following command should compile the `sample1` program above, assuming it was saved to a file called `sample1.F90`. Note, make sure you use the same compiler you used to build AmberTools in the first place.

```
gfortran -I$AMBERHOME/include -L$AMBERHOME/lib -o sample1 sample1.F90 -lsander
```

This command will create a program called *sample1* that you can run from the command-line. Of course as it is written, the program will require that the files prmtop and inpcrd be present in the current directory. It will initialize the *sander* API, compute the energy, and quit without printing anything. Feel free to experiment with your own modifications to these programs.

17.13.2.4. Using the LES Fortran API

To use the LES functionality, you need to use the `sanderles_api` module instead of `sander_api` and you have to link to the sanderles library instead of the sander library (i.e., change `-lsander` to `-lsanderles` in the above compilation step). Since both libraries define most of the same symbols, you unfortunately cannot link both libraries to the same program. For example:

```
gfortran -I$AMBERHOME/include -L$AMBERHOME/lib -o sample1 sample1.F90 -lsanderles
```

17.13.3. The C and C++ APIs

This section describes how to use the C and C++ APIs. These two APIs are the same, and operate very much like a prototypical C API. This is because C and Fortran are both procedural languages (as opposed to object-oriented, like C++). Therefore, Fortran functionality maps more completely onto C than it does onto C++.

The function prototypes and data structures used for the C and C++ APIs are defined in the `sander.h` header file that is installed to `$AMBERHOME/include` when you build AmberTools.

17.13.3.1. Data Structures

The C and C++ data structures are all different `struct`s. Variables within a `struct` are accessed using the . operator.

The sander input options are available as the type `sander_input`, the QM/MM input options are available as the type `qmmm_input_options`, and the potential energy data structure is available as the type `pot_ene`.

An example of using the `sander_input` type is shown in the small code fragment below.

```
#include "sander.h"
sander_input inp;
inp.cut = 9999.0;
inp.ifqnt = 0;
inp.igb = 5;
```

17.13.3.2. Function call syntax

This section details the function calls for the various functions defined in the `sander.h` header file. The syntax is almost identical to the Fortran syntax, except that errors codes are typically returned by the function rather than set in the final input parameter.

```
void gas_sander_input(sander_input *inp, int igb)
```

Unlike the Fortran API, the GB parameter is not optional. This subroutine takes a pointer to a `sander_input` instance and the GB model you wish to use (0 or 6 for vacuum). If an illegal igb value is provided, a warning is printed to `stderr` and a value of 6 is given to `inp->igb`. See table 17.1 for the default values assigned to each variable.

```
void pme_sander_input(sander_input *inp)
```

This subroutine takes a pointer to a `sander_input` instance and initializes every variable inside with the value listed in table 17.1.

```
void qm_sander_input(qmmm_input_options *inp)
```

This subroutine takes a `qmmm_input_options` instance and initializes all of the variables to the values given in Subsection 17.13.1.1. This is the recommended way to initialize the QM/MM options type.

```
int sander_setup_mm(const char* top, double *crd,
                    double *box, sander_input *inp)
```

This function sets up *sander* with the given topology file name, given coordinates, given box dimensions, and input options. Since overloading is not permitted in C, the QM/MM input `struct` cannot be made optional. Therefore, this function can only be used when `inp->ifqnt` is 0. This function returns 0 upon success or 1 upon failure. A system is only considered set up if this function returns 0.

No checking is done to make sure that the number of coordinates provided is correct compared to the number of atoms defined in the topology file. Note that answers will be ridiculous if the coordinate order does not match the atom order in the topology file. Segfaults and other memory violations are possible if the provided coordinate array or box array are too small.

The box array is given in the format $(a,b,c,\alpha,\beta,\gamma)$, which is the same as the format used at the bottom of the input coordinate and restart files. This argument is required even if the system is not periodic, but the values are not used (so they can be initialized to anything).

This function should never be called if a system is already set up.

```
int sander_setup(const char* top, double *crd,
                 double *box, sander_input *inp,
                 qmmm_input_options *qm_inp)
```

This function does the same thing as `sander_setup_mm` described above, but it also requires a pointer to a `qmmm_input_options` instance. If `inp->ifqnt` is set to 0, the contents of `qm_inp` are ignored and a standard MM system is set up. If successful, this function returns 0. Otherwise, it returns 1.

This function should never be called if a system is already set up.

```
void set_positions(double *crd)
```

This function sets the current positions of the active system (and so can only be called if a system is currently set up). Note that the onus is on the programmer to make sure that the coordinate array is large enough. No error checking is done. The input parameter is guaranteed not to change.

```
void set_box(double a, double b, double c,
             double alpha, double beta, double gamma)
```

This function sets the box dimensions and angles from the input parameters (which are guaranteed not to change).

```
void get_positions(double *positions)
```

This function gets the "active" positions for the system that is currently set up.

```
void energy_forces(pot_ene *ener, double *forces)
```

This function will compute the energies and forces from the current conformation of the system that is currently set up and populate the `ener` type and `forces` array with the resulting values. Those parameters are purely output. The energies are all given in units of kilocalories per mole and forces are given in $kcal/mol/\mathring{A}$. This subroutine can only be called if a system is currently set up.

```
void sander_cleanup(void)
```

This function will deallocate all memory used by the *sander* API and return it to a state where no system is set up and a new one can be initialized.

```
int is_setup(void)
```

This function can be called to query whether there is currently a system set up for the *sander* API. It returns 0 if no system is set up and 1 if a system is set up.

```
int sander_natom(void)
```

This funcion will query the currently set up system and return the number of atoms defined by the topology file used during setup. If no system is set up, this function returns 0.

```
int get_inpcrd_natom(const char *filename)
```

This function will open the specified file and try to read how many atoms are defined in that coordinate file. It supports both NetCDF and standard ASCII-formatted inpcrd and restart files. If there was an error in reading the file—either because the file does not exist, read permissions are not set, or the format is unrecognized—the return value will be -1. Otherwise, this function returns the number of atoms defined in the inpcrd file, `filename`.

```
int read_inpcrd_file(const char* filename, double *crd, double *box)
```

This subroutine will read the specified coordinate file and fill the `crd` and `box` arrays with the coordinates and box defined in the file. Both NetCDF restart files and ASCII restart files are supported. If no box is defined in the specified file, the `box` array is initialized to 0. The coordinate array is expected to be allocated with the appropriate amount of space. You can call `get_inpcrd_natom` (described above) to determine how large the coordinate array must be.

If there is a problem reading the file—either because the file does not exist, read permissions are not set, or the format is unrecognized—this function will return 1 and the `crd` array will be uninitialized (the `box` array will still be set to 0). If reading succeeded, this function will return 0. This function can be called regardless of whether a system is currently set up or not.

17.13.3.3. Examples and uses of the C and C++ APIs

In this section, we show examples of how to use the C and C++ API. These samples do exactly the same thing as the two examples in Subsection 17.13.2.3. At the end of this section, we show how to compile your C or C++ program.

We do not do any error checking in these programs since it adds considerably to the length of the example programs. However, you are encouraged to make use of the error reporting in your own programs to avoid program crashes. Syntax highlighting is applied to make the code easier to read.

The first example we provide below shows a sample C program that computes purely MM energies for a non-periodic system using one of the GB models available in Amber.

```c
#include <stdlib.h>
#include "sander.h"
int main() {
    sander_input inp;
    double *crd, *frc;
    double box[6];
    pot_ene ene;
    int natom, ierr;
    // Find out how many atoms are in our inpcrd file
    natom = get_inpcrd_natom("inpcrd");
    crd = (double*) malloc(natom*3*sizeof(double));
    ierr = read_inpcrd_file("inpcrd", crd, box);
    // Set up input options to use igb=5 with 0.2M salt
    gas_sander_input(&inp, 5);
    inp.saltcon = 0.2;
    // Set up our system
    ierr = sander_setup_mm("prmtop", crd, box, &inp);
    // Coordinate array is no longer needed
    free(crd);
    // Find out how big our force array must be
    frc = (double*) malloc(sander_natom()*3*sizeof(double));
    energy_forces(&ene, frc);
    /* Do whatever you want with the energies and forces
     * ...
     * Free up our memory
     */
    sander_cleanup();
    free(frc);
    return 0;
}
```

The second example we provide shows how to use the C API to compute the energy for a periodic system using a multiscale QM/MM Hamiltonian (in a C++ program this time). We will treat residues 10, 11, 12, and 20 using the

PDDG-PM3 Hamiltonian.

```
#include "sander.h"
#include <cstring>
int main() {
    sander_input inp;
    double *crd, *frc;
    double box[6];
    pot_ene ene;
    int natom, ierr;
    // Find out how many atoms are in our inpcrd file
    natom = get_inpcrd_natom("inpcrd");
    crd = new double[natom*3];
    ierr = read_inpcrd_file("inpcrd", crd, box);
    // Set up input options to use igb=5 with 0.2M salt
    pme_sander_input(&inp);
    inp.cut = 10.0;
    qm_sander_input(&qm_inp);
    strncpy(qm_inp.qmmask, “:10-12,20”, 9);
    strncpy(qm_inp.qm_theory, “PDDG-PM3”, 8);
    // Set up our system
    ierr = sander_setup("prmtop", crd, box, &inp, &qm_inp);
    // Coordinate array is no longer needed
    delete[] crd;
    // Find out how big our force array must be
    frc = new double[sander_natom()*3];
    energy_forces(&ene, frc);
    /* Do whatever you want with the energies and forces
     * ...
     * Free up our memory
     */
    sander_cleanup();
    delete[] frc;
    return 0;
}
```

To compile C or C++ programs using the *sander* API, the compiler must be able to find the `sander.h` header file, which are deposited in `$AMBERHOME/include` when you build AmberTools. You must also link `libsander.so` (or `libsanderles.so`) when you link your program. On Mac OS X, these shared libraries are named `libsander.dylib` and `libsanderles.dylib` instead.

The programs we've written above are simple enough that they can be compiled and linked at the same time. The following command should compile the `sample1` program above, assuming it was saved to a file called `sample1.F90`. Note, make sure you use the same compiler you used to build AmberTools in the first place.

```
gcc -I$AMBERHOME/include -L$AMBERHOME/lib -o sample1 sample1.c -lsander
```

This command will create a program called *sample1* that you can run from the command-line. Of course as it is written, the program will require that the files prmtop and inpcrd be present in the current directory. It will initialize the *sander* API, compute the energy, and quit without printing anything. Feel free to experiment with your own modifications to these programs. For the second sample, you need to use a C++ compiler instead of the C compiler.

17.13.3.4. Using the LES C/C++ API

There is only one header file for the *sander* C/C++ API. The LES and standard functionalities are differentiated using the LES preprocessor directive. To use the LES functionality, you need to define the LES macro. You can either do this in the source code (by putting "`#define LES 1`" before `#include "sander.h"`) or by compiling with the `-DLES` flag. If you use the `LES` symbol (either as a variable or a preprocessor macro), you will have to implement this in the source code and undefine the macro after `sander.h` is included. For example, on the command line this would look like:

```
gcc -DLES -I$AMBERHOME/include -L$AMBERHOME/lib -o sample1 sample1.c -lsanderles
```

17.13.4. The Python API

This section describes how to use the Python API so that you can use *sander* functionality inside your own Python scripts. Building the Python bindings requires that the Python development headers and libraries be installed. As long as you install the recommended packages listed on http://ambermd.org/ubuntu.html for your Linux distribution, the necessary prerequisites will be installed.

The *sander* functionality is implemented in the `sander` Python module. The *sander* LES functionality is implemented in the `sanderles` module. While the Python API implements the functions described on page 350, the semantics of how these functions are used in Python differs more than the difference between the C/C++ and Fortran APIs.

The Python API has numerous advantages over the other options. First, processing strings is handled correctly by the boilerplate that interfaces Python with C, meaning that the programmer does not have to worry about how strings map to the underlying Fortran code. Second, data is always initialized, so the programmer does not have to worry about bugs arising from unitiialized variables. Finally, array sizes are determined automatically and no allocation or deallocation is required.

Furthermore, the Python API provided here interacts with other Python packages provided as part of AmberTools—specifically several of the classes provided by ParmEd. See Section 14.2 for more information (specifically Subsection 14.2.5.3 for the ParmEd Python API documentation).

17.13.4.1. Data Structures

The data structures in the Python API are all "restricted" classes. I say 'restricted' because setting new attributes is not supported and will raise an `AttributeError`. The data types for the *sander* input options, QM/MM input options, and potential energy terms are the classes `InputOptions`, `QmInputOptions`, and `EnergyTerms`, respectively. The last class is part of the private `sander._pys` namespace since it is only produced as output and never needed as input, whereas the first two are members of the `sander` package namespace.

Unlike C and Fortran, the Python classes have default constructors that will initialize all of the variables for the different classes. An example of using the `InputOptions` class is shown below.

```
import sander
inp = sander.InputOptions()
inp.cut = 9999.0
inp.extdiel = 78.5
inp.intdiel = 1
```

17.13.4.2. Function call syntax

This section details the function calls for the various functions defined in the `sander` package.

```
inp = sander.gas_input(igb=6)
```

The `igb` argument is an optional integer that defaults to 6 (vacuum). This function returns an initialized `InputOptions` instance whose values are listed in table 17.1. If an illegal igb value is provided, a `ValueError` is raised.

```
inp = sander.pme_input()
```

This subroutine returns a `InputOptions` instance and initializes every member with the value listed in table 17.1.

```
qm_inp = sander.qm_input()
qm_inp = sander.QmInputOptions()
```

These two commands both return a `QmInputOptions` instance and initializes all of the variables to the values given in Subsection 17.13.1.1. The function (`sander.qm_input`) is redundant, since the `QmInputOptions` constructor does the same thing. The function was provided only for consistency with the Fortran and C/C++ APIs.

```
sander.setup(prmtop, coordinates, box, mm_options, qm_options=None)
```

This function sets up *sander* with the given topology file, coordinates, box dimensions, and input options. If `mm_options.ifqnt` is 1 and `qm_options` is not provided, a `ValueError` is raised. The topology file can be either an `AmberParm` instance (see Subsection 14.2.5.3 for more information) or a string filename pointing to a valid Amber topology file.

The coordinate array can either be a `numpy.ndarray` instance an `array.array` instance, or a `list`. The array must be 1-dimensional with a length equal to 3 × natom. In particular, the coordinate array taken from a `Rst7` instance can be used. Alternatively, the `coordinates` argument can be a string that is the filename of a coordinate or restart file.

The box array, too, can be a `numpy.ndarray`, `array.array`, or `list` instance of length 6. If it is not one of those data types, a `TypeError` will be raised. If it does not have 6 elements, a `ValueError` will be raised. If no box is needed, the `box` argument can be set to `None`. Alternatively, if `box` is set to `None` and a filename was passed to the `coordinates` argument that contains box dimensions, the box will be set from the information in that file. However, any box dimensions passed using the `box` argument will take precedence.

The `mm_options` must be a `InputOptions` instance or a TypeError will be raised. The `qm_options` must be a `QmInputOptions` instance or `None`. Otherwise, a `TypeError` will be raised.

If there is any problem setting the system up, or if a system is already set up, a `RuntimeError` will be raised. This "function" is actually a class that implements the context manager protocol via the `with` statement (Python 2.5 or greater, only—Python 2.4 users must use the syntax above).

```
with sander.setup(prmtop, coordinates, box, mm_options, qm_options=None):
    ... do stuff
```

The return value of `sander.setup` is a reference to the class (which itself can be used in a context manager). Upon exiting the context manager, `sander.cleanup` is called (but only if `sander.setup` succeeded).

```
sander.set_positions(crd)
```

This function sets the current positions of the active system. The `crd` argument can be a `numpy.ndarray`, `array.array`, or `list` instance and must be either 1-dimensional with a length 3 × natom or 2-dimensional with a shape of natom, 3. If the array is not the correct length, a `ValueError` will be raised. If it is not one of the aforementioned types, a `TypeError` will be raised. If a system is not currently set up, a `RuntimeError` will be raised. This function returns `None`.

```
sander.set_box(a, b, c, alpha, beta, gamma)
```

This function sets the box dimensions and angles from the input parameters. If the incorrect number of arguments are given, or if the arguments are not all numbers, a `TypeError` is raised. If no system is currently set up, a `RuntimeError` is raised. This function returns `None`.

```
positions = sander.get_positions()
```

This function returns the coordinates as a one-dimensional list for the currently active system. If no system is currently set up, a `RuntimeError` is raised.

```
ene, frc = sander.energy_forces()
```

This function will compute the energies and forces from the current conformation of the system that is currently set up and returns a two-element `tuple` in which the first element is an `EnergyTerms` instance with the attributes listed in Subsection 17.13.1.1 and the second attribute is a $3 \times$ natom-length `list` with the atomic forces. The energies are all given in units of kilocalories per mole and forces are given in $kcal/mol/Å$. A `RuntimeError` is raised if no system is currently set up.

```
sander.cleanup()
```

This function will deallocate all memory used by the *sander* API and return it to a state where no system is set up and a new one can be initialized. If no system is set up, a `RuntimeError` is raised. This function returns `None`.

```
bool = sander.is_setup()
```

This function can be called to query whether there is currently a system set up for the *sander* API. It returns `False` if no system is set up and `True` if a system is set up.

```
natom = sander.natom()
```

This function will query the currently set up system and return the number of atoms defined by the topology file used during setup. If no system is set up, this function raises a `RuntimeError`.

Coordinate file parsing No functions are provided to parse and query coordinate and restart files, since the `Rst7` class from the chemistry.amber package already does that. Examples using this class are shown in the next section.

17.13.4.3. Examples and uses of the Python API

In this section, we show examples of how to use the Python API. These samples do exactly the same thing as the two examples in Subsection 17.13.2.3 and Subsection 17.13.3.3.

Unlike the previous APIs, the Python API has built-in error checking through the utilization of the Exception mechanism. The various exceptions that can be raised and the circumstances in which they will be raised are described in the previous section. You may wish to catch some of the exceptions in your own Python scripts to implement more elaborate error handling. Notice that the Python program here is much simpler than the equivalent Fortran and C programs presented earlier.

These programs also make use of the `AmberParm` class in the chemistry package that is part of the ParmEd program (see Section 14.2).

```
import sander
from chemistry.amber.readparm import AmberParm
# Initialize the topology object with coordinates
parm = AmberParm("prmtop", "inpcrd")
# Set up input options to use igb=5 with 0.2M salt
inp = sander.gas_input(igb=5)
inp.saltcon = 0.2
sander.setup(parm, parm.coordinates,None, inp)
# Compute the energies and forces
ene, frc = sander.energy_forces()
# Do whatever you want with the energies and forces
# ...
# Free up our memory
sander.cleanup()
```

The second example we provide shows how to use the Python API to compute the energy for a periodic system using a multiscale QM/MM Hamiltonian. We will treat residues 10, 11, 12, and 20 using the PDDG-PM3 Hamiltonian. Also, rather than loading the inpcrd file directly into the AmberParm object, we use the `open` constructor of the `Rst7` class to read in the coordinate file. While this is exactly what the `AmberParm` class does under the hood, this approach is presented here to show how to use the `Rst7` class in your own programs.

```
import sander
from chemistry.amber.readparm import AmberParm, Rst7
# Initialize the topology object with coordinates
parm = AmberParm("prmtop")
rst = Rst7.open("inpcrd")
# Set up input options to use PME with a 10A cutoff
inp = sander.gas_input(igb=5)
inp.cut = 10.0
qm_inp = sander.QmInputOptions()
qm_inp.qmmask = ":10-12,20"
qm_inp.qm_theory = "PDDG-PM3"
sander.setup(parm, rst.coords, rst.box, inp, qm_inp)
# Compute the energies and forces
ene, frc = sander.energy_forces()
# Do whatever you want with the energies and forces
# ...
# Free up our memory
sander.cleanup()
```

One final thing we will mention is that the sander Python API supports the context manager protocol! The previous example can be rewritten as

```
import sander
from chemistry.amber.readparm import AmberParm, Rst7
# Initialize the topology object with coordinates
parm = AmberParm("prmtop")
rst = Rst7.open("inpcrd")
# Set up input options to use PME with a 10A cutoff
inp = sander.gas_input(igb=5)
inp.cut = 10.0
qm_inp = sander.QmInputOptions()
qm_inp.qmmask = ":10-12,20"
qm_inp.qm_theory = "PDDG-PM3"
with sander.setup(parm, rst.coords, rst.box, inp, qm_inp):
    # Compute the energies and forces
    ene, frc = sander.energy_forces()
# Do whatever you want with the energies and forces
# ...
# Free up our memory
```

When the context manager is exited (i.e., when program execution is no longer inside the `with` block), sander is automatically cleaned up. This occurs regardless of whether or not an error was raised during the execution of the code within the `with` block. Notice how `sander.cleanup()` is no longer necessary.

17.13.4.4. Using the LES Python API

To use the LES functionality in Python, you need to import the `sanderles` package instead of the `sander` package. Note that while nothing stops you from importing both the `sander` and `sanderles` packages in the same Python script, both packages will not work correctly in the same script.

18. pmemd

18.1. Introduction

PMEMD (Particle Mesh Ewald Molecular Dynamics) was initially a reimplementation of a subset of sander functionality, although it has since diverged somewhat into the primary molecular dynamics engine within the AMBER Software suite, written with the primary goal of improving the performance of the most frequently used methods of sander. PMEMD supports Particle Mesh Ewald simulations, Generalized Born simulations, Isotropic Periodic Sum, ALPB (Analytical Linearized Poisson-Boltzmann) and as of AMBER v16 gas phase simulations using both the AMBER and CHARMM Force fields.

One of the major additions to PMEMD in recent years has been addition of support for acceleration of PME, GB and IPS calculations using NVIDIA GPUs.[378–380] A detailed overview is provided in section 18.6

For the supported functionality, the input required and output produced are intended to exactly replicate sander within the limits of machine roundoff differences. PMEMD simply runs more rapidly, scales better in parallel using MPI, can be used profitably on significantly higher numbers of processors, can make use of NVIDIA GPUs and Intel Xeon Phis for acceleration and uses less resident memory. Dynamic memory allocation is used so memory configuration is not required. PMEMD is ideal for molecular dynamics simulations of both small and large solvated systems for long periods of time, especially if supercomputer or GPU resources are available. Benchmark data is available on the Amber website, ambermd.org. Given the improvements in performance in both serial and parallel as well as the incredible performance offered by GPU acceleration it is advisable to always use PMEMD in place of sander if the simulation requirements are within the functionality envelope provided by PMEMD.

PMEMD accepts sander input files (*mdin, prmtop, inpcrd, refc*). All options documented in the sander section of this manual should be properly parsed and an error message generated if a requested feature is not supported. PMEMD is also backward compatible in regard to input to the same extent as sander.

18.2. Functionality

New functionality has been added to *pmemd* in release 16 of Amber including:

- Support for gas phase simulations (through igb=6) [cpu & gpu]
- Support for external electric fields [cpu]
- Support for the Charmm VDW Force switch [cpu & gpu]
- Support for Gaussian accelerated molecular dynamics [cpu & gpu]

and the following functionality, as of AMBER 16, is also now supported by the GPU version of pmemd in addition to the CPU version:

- Semi-Isotropic pressure scaling
- Enhanced NMR restraints and R^6 averaging support
- Expanded umbrella sampling support
- Constant pH and REMD Constant pH

In addition the following will soon be added via AMBER's update mechanism during compilation:

- Support for Adaptively biased MD

- Support for the upcoming Knights Landing Intel Xeon Phi on day of release
- Improved CPU performance and scaling
- Thermodynamic Integration, FEP and MBAR support on GPUs

As mentioned above, PMEMD is not a complete implementation of sander. Instead, it is intended to be a fast implementation of the functionality most likely to be used by someone doing long time scale explicitly or implicitly solvated systems. It also includes some additional functionality of it's own.

Specifically the following functionality in sander is missing entirely:

imin=5 In &cntrl. Trajectory analysis is not supported.

nmropt=2 In &cntrl. A variety of NMR-specific options such as NOESY restraints, chemical shift restraints, pseudocontact restraints, and direct dipolar coupling restraints are not supported.

idecomp!=0 In &cntrl. Energy decomposition options, used in conjunction with mm_pbsa, are not supported.

ipol!=0 In &cntrl. Polarizable force field simulations are not supported.

igb==10 In &cntrl. Poisson-Boltzmann simulations are not supported.

ntmin>2 In &cntrl. XMIN and LMOD minimization methods are not supported.

Solvent Caps Solvent cap simulations are not supported.

itgtmd!=0 In &cntrl. Targeted molecular dynamics is not supported.

ievb!=0 In &cntrl. Empirical Valence Bond methods are not supported.

ifqnt!=0 In &cntrl. QM/MM methods are not supported.

&debugf namelist Use of the &debugf namelist is only supported in a very limited way. Specifically only the `do_charmm_dump_gold` option is supported.

ineb!=0 In &cntrl. Nudged elastic band (NEB) calculations are not supported. These calculations are done by sander.MPI.

LES The Locally Enhanced Sampling method is not supported.

REM==2 The partial REMD method (for LES) is not supported

iamoeba!=0 In &cntrl. The amoeba polarizable potentials of Ren and Ponder are not supported in pmemd, although support is provided through a special pmemd.amoeba implementation provided as part of Amber 16.

The following &ewald options are supported, but only with the indicated default values:

ew_type=0 Only Particle Mesh Ewald calculations are supported. ew_type = 1 (regular Ewald calculations) must be done in sander.

nbflag=1 The nbflag option is ignored for MD, and all nonbonded list updates are scheduled based on "skin" checks. This is more reliable and has little cost. The variable nsnb still can be set and has an influence on minimizations but is ignored during MD. For PME calculations, list building may also be scheduled based on heuristics to suit load balancing requirements in multiprocessor runs.

nbtell=0 The nbtell option is not particularly useful and is ignored.

eedmeth=1 Only a cubic spline switch function (eedmeth = 1) for the direct sum Coulomb interaction is supported. This is the default, and most widely used setting for eedmeth. On some machine architectures, energies and forces are actually splined as a function of r**2 to a higher precision than the cubic spline switch.

column_fft=0 This is a sander-specific performance optimization option. PMEMD uses different mechanisms to enhance performance, and ignores this option.

It is suggested that new PMEMD users simply take an existing sander mdin file and attempt a short 10-30 step run. The output will indicate whether or not PMEMD will handle the particular problem at hand for all the functionality that is supported by "standard" sander. For functionality that requires special builds of sander or sander-derived executables (LES), there may be failures in namelist parsing.

18.3. PMEMD-specific namelist variables

The following namelist options are specific to PMEMD and generally relate to PMEMD specific performance optimizations: default values:

mdout_flush_interval In &cntrl, this variable can be used to control the minimum time in integer seconds between "flushes" of the mdout file. PMEMD DOES NOT use file flush() calls at all because flush functionality does not work for all fortran compilers used in building pmemd. Thus, pmemd does an open/close cycle on mdout at a default minimum interval of 300 seconds. This interval can be changed with this variable if desired in the range of 0-3600. If *mdout_flush_interval* is set to 0, then mdout will be reopened and closed for each printed step. This functionality is provided in pmemd because some large systems have such large file i/o buffers that mdout will have 0 length on the disk through 100's of psec of simulated time. The default of 300 seconds provides a good compromise between efficiency and being able to observe the progress of the simulation.

mdinfo_flush_interval In &cntrl, this variable can be used to control the minimum time in integer seconds between "flushes" of the mdinfo file. PMEMD DOES NOT use file flush() calls at all because flush functionality does not work for all fortran compilers used in building pmemd. Thus, pmemd does an open/close cycle on mdinfo at a default minimum interval of 60 seconds. This interval can be changed with this variable if desired in the range of 0-3600. Note that mdinfo under pmemd simply serves as a heartbeat for the simulation at *mdinfo_flush_interval*, and mdinfo probably will not be updated with the last step data at the end of a run. If *mdinfo_flush_interval* is set to 0, then mdinfo will be reopened and closed for each printed step.

es_cutoff, vdw_cutoff In &cntrl, these variables can be used to control the cutoffs used for vdw and electrostatic direct force interactions in PME calculations separately. If you specify these variables, you should not specify the cut variable, and there is a requirement that *vdw_cutoff* >= *es_cutoff*. These were introduced anticipating the need to support force fields where the direct force calculations are more expensive. For the current force fields, one can get slightly improved performance and about the same accuracy as one would get using a single cutoff. A good example would be using *vdw_cutoff*=9.0, *es_cutoff*=8.0. For this scenario, one gets about the accuracy in calculations associated with 9.0 angstrom cutoffs, but at a cost intermediate between an 8.0 and a 9.0 angstrom cutoff.

no_intermolecular_bonds In &cntrl. New variable controlling molecule definition. If 1, any molecules (ie., molecules as defined by the prmtop) joined by a covalent bond are fused to form a single molecule for purposes of pressure and virial-related operations; if 0 then the old behaviour (use prmtop molecule definitions) pertains. The default is 1; a value of 0 is not supported with forcefields using extra points. This option was necessitated in order to efficiently parallelize model systems with extra points. This redefinition of molecules actually allows for a more correct treatment of molecules during pressure adjustments and should produce better results with less strain on covalent bonds joining prmtop-defined molecules, but if the default value is used for a NTP simulation, results will differ slightly relative to sander if any intermolecular bonding was applied in forming the prmtop (eg., a cyx-cyx bridge was added between two peptides that originated in a PDB file, with each peptide having its own "TER" card). If consistency with sander is more important to you, and you are not using extra points, then you may want to set *no_intermolecular_bonds* to 0.

ene_avg_sampling In &cntrl. New variable controlling the number of steps between energy samples used in energy averages. If not specified, then ntpr is used (default). To match the behaviour of sander or PMEMD v9 or earlier, this variable should be set to 1. This variable is only used for MD, not minimization and will also effectively be turned off if *ntave* is in use (non-0) or RESPA is in use (nrespa > 1). It is a fairly common situation that it is completely unnecessary to sample the energies every step to get a good average during production, and this is costly in terms of performance. Thus, performance can be improved (with greatest improvements for the ensembles in the order NVE > NVT > NTP) without really losing anything of value by using the new default for energy average sampling (specify nothing).

use_axis_opt In &ewald. For parallel runs, the most favorable orientation of an orthogonal unit cell is with the longest side in the Z direction. Starting with pmemd 3.00, internal coordinates were actually reoriented to take advantage of this, and in high processor count runs on oblong unit cells, using axis optimization can improve performance on the order of 10%. However, if a system has hotspots, the results produced with axes oriented differently may vary by on the order of 0.05% relatively quickly. This effect has to do with the fact that axis optimization changes the order of LOTS of operations and also the fft slab layout, and under mpi if the system has serious hotspots, shake will come up with slightly different coordinate sets. This is really only a problem in pathological situations, and then it is probably mostly telling you that the situation is pathological, and neither set of results is more correct (typically the ewald error term is also high). In routine regression testing with over a dozen tests, axis reorientation has no effect on results. Nonetheless, the defaults are now selected to be in favor of higher reproducibility of results. Axis optimization is only done for mpi runs in which an orthogonal unit cell has an aspect ratio of at least 3 to 2. It is turned off for all minimization runs and for runs in which velocities are randomized (ntt = 2 or 3). If you want to force axis optimization, you may set *use_axis_opt* = 1 in the &ewald namelist. If you set it to 0, you will force it off in scenarios where it would otherwise be used.

fft_grids_per_ang In &ewald. This variable may be used to set the desired reciprocal space fft grid density in terms of fft grids/angstrom. The nearest grid dimensions, given the prime factors supported by the underlying fft implementation, that meet or exceed this density will be used (ie., *nfft1,2,3* are set based on this specification). The default value is 1.0 grids/angstrom and gives very reasonable accuracy. PMEMD is actually more stringent now than sander in that it will meet or exceed the desired density instead of just approximating it. Thus, to get identical results with sander, one may have to specify grid dimensions to be used with the *nfft1,2,3* variables.

18.4. Slightly changed functionality

An I/O optimization has been introduced into PMEMD. If the user does not specify a value of NTWR then it defaults to NSTLIM. In general, frequent writes of restrt, especially in runs with a high processor count or on GPUs, is wasteful. Also, if the mden file is being written, it is always written as formatted output, regardless of the value of *ioutfm*. SANDER now conforms to this convention regarding *ioutfm* and mden.

In thermodynamic integration calculations, the input format is different from SANDER. The differences are explained in section 21.1 of the manual.

In addition, there are two command-line options unique to pmemd:

-l <logfile name> A name may be assigned to the log file on the command line.

-gpes <process_map_file> This option controls the distribution of threads in a *multipmemd* simulation and allows you to allocate threads to various processes however you want (rather than dividing up all threads equally between each line of the groupfile). By default, the threads are allocated sequentially (that is, for N groups given M threads per group, or N*M threads total, threads 0 to M-1 will be assigned to the first group, threads M to 2M-1 will be assigned to the second group, etc.). Using the -ng-nonsequential flag, threads will be allocated with a one-at-a-time approach. For instance, given the same setup as above, the first group gets threads 0, M, 2M, 3M, ... etc, the second group gets threads 1, M+1, 2M+1, 3M+1, ... etc.

The process_map_file is a file that contains as many lines as you have groups (although the last line can be omitted, and the remaining unspecified threads will be assigned to the final group). Each line must contain space-delimited integers that correspond to the thread numbers you want assigned to that group. Each group is assigned the threads listed in that line of the process_map_file. Every thread must be specified in the process_map_file, and no thread can be specified more than once.

18.5. Parallel performance tuning and hints

In order to achieve higher scaling, pmemd has implemented several performance algorithms, the most notable of which is the option of using a "block" or pencil fft rather than the usual slab fft algorithm. The block fft algorithm allows the reciprocal space and fft workload to be distributed to more processors, but at a cost of higher communications overhead, both in terms of the distributed fft transpose cost and in terms of communication of the data necessary to set up the fft grids in the first place. A number of variables in the &ewald namelist can be used to control whether the slab or block fft algorithm is used, how the block division occurs, whether direct force work is also assigned to tasks doing reciprocal space and fft work, whether the master is given any force and energy computation work to do, as opposed to being reserved strictly for handling output and loadbalancing, and the frequency of atom ownership reassignment, an operation that counteracts rising communications costs caused by diffusion. The various namelist variables involved have all been assigned defaults that adapt to run conditions, and in general it is probably best that the user just use the defaults and not attempt to make adjustments. However, in some instances, fine tuning may yield slightly better performance. The variables involved include *block_fft, fft_blk_y_divisor, excl_recip, excl_master,* and *atm_redist_freq*. These are described further in the README under pmemd/src as well as in the sourcecode itself.

Performance depends not only on proper setup of hardware and software, but also on making good choices in simulation configuration. There are many tradeoffs between accuracy and cost, as one might expect, and understanding all of these comes with experience. However, I would like to suggest a couple of good choices for your simulations, if you have facilities where you can routinely run at high processor count, say 32 processors or more. First of all, there is an implementation of binary trajectory files in pmemd and sander, based on the netCDF binary file format. This is invoked now using *ioutfm* == 1, assuming you have built either pmemd or sander with "bintraj" support. Using this output format, i/o from the master process will be more efficient and your filesize will be about half what it would otherwise be. In Amber, ptraj can read these new netCDF trajectory files and can convert them to ASCII format if needed. At really high processor count using the netCDF format can be on the order of 10% more efficient than using the standard formatted trajectory output. Secondly, other simulation packages typically use multiple timestepping (respa) methods as an efficiency measure. These methods typically sample reciprocal space forces for PME less frequently. Due to the limited use of such methods by Amber users this approach has not been optimized in pmemd and hence while this can slightly improve performance for pmemd at low processor count, at higher processor counts using respa typically makes loadbalancing less efficient leading to a net loss of performance. If you wish to use respa for pme simulations (done typically by setting *nrespa* to 2 or 4), then you should check whether you actually get better performance. You may well not, and it will be at a cost of a loss in accuracy. Using respa for generalized Born simulations is fine in all cases, however.

18.6. GPU Accelerated PMEMD

One of the newer features of PMEMD, is the ability to use NVIDIA GPUs to accelerate both explicit solvent PME and implicit solvent GB simulations [378–380]. This work is by Ross Walker at the San Diego Supercomputer Center and Scott Le Grand in collaboration with NVIDIA. While this GPU acceleration is considered to be production ready, and has been heavily used since its release as part of AMBER 11, it is still evolving and has not been tested to the same extent as the CPU code; users should exercise caution when using this code. The error checking is not as verbose in the GPU code as it is on the CPU. In particular, simulation failures, such as atom collisions or other simulation instabilities, will manifest themselves as CUDA launch errors or GPU download failures and not as informative error messages. If you encounter problems during a simulation on the GPU you should first try to run the identical simulation on the CPU to ensure that it is not your simulation setup which is causing problems. Feedback and questions should be posted to the Amber mailing list (see *http://lists.ambermd.org/*).

This section of the manual describes the feature set, installation, performance and accuracy considerations and other aspects of GPUs at the time of Amber's release. However, the rapidly changing nature of this field means that frequent updates are likely. You should refer to the web page *http://ambermd.org/gpus/* for the most up to date information.

18.6.1. Supported Features

The GPU accelerated version of PMEMD supports both explicit solvent PME simulations in all three canonical ensembles (NVE, NVT and NPT) and implicit solvent Generalized Born simulations. It has been designed to support as many of the standard PMEMD features as possible, however, there are some current limitations that are detailed below. Some of these may be addressed in the future, and patches released, with the most up to date list posted on the web page. The following options are **NOT** supported:

1. *ibelly* $\neq$ *0* Simulations using belly style constraints are not supported.
2. *icfe* $\neq$ 0 Support for TI is not currently implemented (but an update to support this in AMBER 16 is planned).
3. *igb* $\neq$ *0* && *cut* < *systemsize* GPU accelerated implicit solvent GB simulations do not support a cutoff.
4. *nmropt* > *1* Support is not currently available for nmropt > 1. In addition for nmropt = 1 only features that do not change the underlying force field parameters are supported. For example umbrella sampling restraints are supported as is Jarzynski sampling as well as simulated annealing functions such as variation of Temp0 with simulation step. However, varying the VDW parameters with step is not supported.
5. *nrespa* $\neq$ *1* No multiple time stepping is supported.
6. *vlimit* $\neq$ *-1* For performance reasons the vlimit function is not implemented on GPUs.
7. *imin* = *1 with MPI* Minimization is currently only supported for single GPU runs.
8. *es_cutoff* $\neq$ *vdw_cutoff* Independent cutoffs for electrostatics and van der Waals are not supported on GPUs.
9. *order* > *4* PME interpolation orders of greater than 4 are not supported at present.
10. *emil_do_calc* $\neq$ 0 Emil is not supported on GPUs.
11. *lj1264* $\neq$ 0 The 12-6-4 potential is not supported on GPUs.
12. *iemap* > 0 EMAP restraints are not supported on GPUs.
13. *isgld* > 0 Self guided langevin dynamics are not supported on GPUs.

Additionally there are some minor differences in the output format. For example the Ewald error estimate is *not* calculated when running on a GPU. It is recommended that you first run a short simulation using the CPU code to check that the Ewald error estimate is reasonable and that your system is stable. The above limitations are tested for in the code; however, it is possible that there are additional simulation features that have not been implemented or tested on GPUs.

18.6.2. New in Amber 16

Amber 16 represents a continued evolution of the Amber GPU MD ecosystem with a number of additions to functionality, support for new hardware and extensive optimization appearing in the latest release. One of the key advances in Amber 16 is code optimization that provides a speed boost of approximately 30% on the current Maxwell generation (M40, M60, GTX980, GTX980TI and GTX-Titan-X) hardware. At the time of writing a single node containing 4 GTX-Titan-X GPUs achieves over 550ns/day for the DHFR 4fs NVE benchmark with an aggregate throughput of over 1.4 microseconds per day possible. This makes AMBER 16 the world's fastest molecular dynamics software package on commodity hardware.

Additional improvements in Amber 16 include:

- Support for semi-isotropic pressure scaling
- Support for the Charmm VDW force switch
- Enhances NMR restraint support and R^6 averaging support
- Gaussian accelerated molecular dynamics
- Expanded umbrella sampling support
- Constant pH and REMD Constant pH support
- Support and significant performance improvements for the latest Maxwell and the soon to be released Pascal GPUs from NVIDIA
- Adaptively biased MD (coming soon via automatic update)
- Thermodynamic Integration, FEP and MBAR (coming soon via an automatic update)

18.6.3. Supported GPUs

GPU accelerated PMEMD has been implemented using CUDA and thus will only run on NVIDIA GPUs at present. Due to accuracy concerns with pure single precision the code uses a custom designed hybrid single/double/fixed-precision model termed SPFP. This places the requirement that the GPU hardware supports both double precision and integer atomics meaning only GPUs with hardware revision 2.0 (C2050 / GTX480) and later can be used. Support for hardware revision 1.3 (C1060 / GTX295) was present in previous versions of the code but for code complexity and maintenance reasons has been deprecated as of AMBER 14. Support is provided for Tesla, Quadro and GeForce GPUs and almost all mid to high end cards are supported. You are encouraged to visit the AMBER website (ambermd.org/gpus/) for an update to date list of supported hardware and performance numbers.

You should ensure that all GPUs on which you plan to run PMEMD are connected to PCI-E 2.0 x16 lane slots or better. For peer to peer communication (required for multi-GPU scaling) the cards to be used need to be in the same PCI-E domain. If this is not the case then you will likely see significantly degraded performance in parallel. For more information, and details of optimium hardware designs. please see Recommended Hardware under the GPU section of the AMBER website (*http://ambermd.org/gpus/*).

Support is provided for single GPU and multiple GPU runs. Employing multiple GPUs in a single simulation requires MPI and the pmemd.cuda.MPI executable. If you have multiple simulations to run then the recommended method is to use one GPU per job. The pmemd GPU code has been developed in such a way that for single GPU runs the PCI-E bus is only used for I/O. This sets AMBER apart from other MD packages since it means the CPU specs do not feature in the GPU code performance. As such low end economic CPUs can be used. Additionally it means that in a system containing 4 GPUs 4 individual calculations can be run at the same time without interfering with each other's performance. Selection of which GPU is used for single GPU runs is automatic if the GPUs are set to process exclusive mode (`nvidia-smi -c 3`) but the recommended approach is to use the CUDA_VISIBLE_DEVICES environment variable to select which GPU should be used. For parallel runs you specify the same number of MPI threads as you have GPUs you want to use. The code will automatically use peer to peer communication if available. Multi-GPU runs require the GPUs to be set to default mode (`nvidia-smi -c 0`). More details are provided in section 18.6.6.

18.6.4. Accuracy Considerations

The nature of current generation GPUs is such that single precision arithmetic is considerably faster (>200x for M40, >23x for K10 and >2x for K20) than double precision arithmetic. This poses an issue when trying to obtain good performance from GPUs. Traditionally the CPU code in Amber has always used double precision throughout the calculation. While this full double precision approach has been implemented in the GPU code it gives very poor performance and so the default precision model used when running on GPUs is a combination of single and fixed precision, termed hybrid precision (SPFP), that is discussed in further detail in references [378–380]. This approach uses single precision for individual calculations within the simulation but fixed scaled integer precision for all accumulations. It also uses fixed precision for shake calculations and for other parts of the code where loss of precision was deemed to be unacceptable. Tests have shown that energy conservation is equivalent to the full double precision code and specific ensemble properties, such as order parameters, match the full double precision CPU code. Previous acceleration approaches, such as the MDGRAPE-accelerated *sander*, have used similar hybrid precision models and thus we believe that this is a reasonable compromise between accuracy and performance. The user should understand though that this approach leads to rapid divergence between GPU and CPU simulations, similar to that observed when running the CPU code across different processor counts in parallel but occuring much more rapidly. Ultimately though this is simply a cosmetic consideration since any statistical mechanical properties should converge to the same values.

While the default precision model is currently the hybrid SPFP model two additional precision models have been implemented within version 16 of the GPU code to facilitate advanced testing and comparison. The choice of default precision model may change in the future based on the outcome of detailed validation tests of the different approaches. The precision models supported, all of which are built automatically at compile time are:

- **SPFP** - (*Default*) Use a combination of single precision for calculation and fixed precision for accumulation. This approach is believed to provide the optimum tradeoff between accuracy and peformance and hence at the time of release is the default model invoked when using the executable *pmemd.cuda*.

- **DPFP** - Use double precision (and double precision equivalent fixed precision) for the entire calculation. This provides for careful regression testing against the CPU code. It makes no additional approximations above and beyond the CPU implementation and would be the model of choice if performance was not a consideration. On v2.0 NVIDIA hardware (e.g. M2090) the performance is approximately half that of the SPFP model while on v3.0 NVIDIA hardware (e.g. K10) the performance is substantially less than the SPFP model.

- **SPXP** - (*Experimenta*l) Use single precision for calculation and a combination of 32 bit integer accumulation strategies to approximate 48 bit precision in the summation stage. This precision model has been designed to provide future proofing of performance on next and later generation hardware designs. It is considered experimental at present and should not be used for production simulations except as a way to test how the model performs.

18.6.5. Installation and Testing

The GPU version of PMEMD is called pmemd.cuda (or pmemd.cuda.MPI for the multi GPU version) and must be built separately from the standard serial and parallel installations. Before attempting to build the GPU version of PMEMD you should have built and tested at least the serial version of Amber and preferably the parallel version as well. This will help to ensure that basic issues relating to standard compilation on your hardware and operating system do not lead to confusion with GPU related compilation and testing problems. You should also be familiar with Amber's compilation and test procedures.

It is assumed that you have already correctly installed and tested CUDA support on your GPU. Additionally the environment variable CUDA_HOME should be set to point to your NVIDIA Toolkit installation and $CUDA_HOME/bin/ should be in your path. At the time of release CUDA 7.5 or later is required with 7.5 having been tested and officially supported.

Building and Testing the GPU code

In a change from previous versions of Amber the GPU compilation and testing procedure has been simplified such that all three precision models are automatically built and the testing procedure has been made more reliable with less false negatives during testing.

Assuming you have a working CUDA installation you can build all three precision models (*pmemd.cuda_SPFP, pmemd.cuda_DPFP* and *pmemd.cuda_SPXP*) with *pmemd.cuda* linked to pmemd.cuda_SPFP as follows:

```
export CUDA_HOME=/usr/local/cuda  (or other appropriate location if not already set in bashrc or cshrc
cd $AMBERHOME
./configure -cuda gnu
make install
```

Next you can run the tests using the default GPU (the one with the largest memory followed by lowest GPU ID) with:

```
make test
```

The majority of these tests should pass. However, given the parallel nature of GPUs, meaning the order of operation is not well defined, it is not uncommon for there to be several possible failures, although substantially less than ith earlier versions of Amber. You may also see some tests, particularly the GB nucleosome test, fail on GPUs with limited memory. You should inspect the diff file created in the *$AMBERHOME/logs/test_amber_cuda/* directory to manually verify any possible failures. Differences which occur on only a few lines and are minor in nature can be safely ignored. Any large differences, or if you are unsure, should be posted to the Amber mailing list for comment.

Testing Alternative GPUs

Should you wish to run the tests on a GPU different from the one with the most memory (and lowest GPU ID if more than one identical GPU exists) then you should make use of the CUDA_VISIBLE_DEVICES environment variable as described below. For example, to test the GPU with ID = 2 (GPUs are numbered sequentially from zero) and the default SPFP precision model you would run the tests as follows:

```
cd $AMBERHOME
export CUDA_VISIBLE_DEVICES=2
make test
```

Building pmemd.cuda.MPI

The GPU version of pmemd can be run in parallel on multiple GPUs using the executable pmemd.cuda.MPI. Some simulations, particularly replica exchange simulations, require a parallel executable in order to operate.

Assuming you have a working CUDA and MPI installation you can build *pmemd.cuda.MPI* as follows:

```
cd $AMBERHOME
./configure -cuda -mpi gnu
make install
```

Next you can run the tests using the default GPUs (the one with the largest memory in descending order) with:

```
export DO_PARALLEL="mpirun -np 2" # for bash/sh
setenv DO_PARALLEL "mpirun -np 2" # for csh/tcsh
make test.cuda_parallel
```

The majority of these tests should pass. However, as described above it is not uncommon for there to be some possible failures. You should inspect the diff file created in the *$AMBERHOME/logs/test_amber_cuda_parallel/* directory to manually verify any possible failures. Differences which occur on only a few lines and are minor in nature can be safely ignored. Any large differences, or if you are unsure, should be posted to the Amber mailing list for comment.

18.6.6. Running GPU Accelerated Simulations

In order to run a GPU accelerated MD simulation the only change required is to use the executable pmemd.cuda in place of pmemd. E.g.

```
$AMBERHOME/bin/pmemd.cuda -O -i mdin -o mdout -p prmtop \
-c inpcrd -r restrt -x mdcrd
```

This will automatically run the calculation on the GPU with the most memory even if that GPU is already in use. If you have only a single CUDA capable GPU in your machine then this is fine; however if you want to control which GPU is used, or you want to run multiple independent simulations using different GPUs, then you manually need to specify the GPU to use with the CUDA_VISIBLE_DEVICES environment variable.

CUDA_VISIBLE_DEVICES Specifies which GPU should be used for running a GPU accelerated PMEMD calculation. This is based on the hardware ID of the GPU card which can be obtained by unsetting the variable (unset CUDA_VISIBLE_DEVICES) and running the deviceQuery command from the NVIDIA CUDA SDK. Valid values are a list of integers from 0 to 32. Multiple GPUs may be listed with commas in between them, and the one with the most memory will be selected. For example:

```
export CUDA_VISIBLE_DEVICES=1,3
$AMBERHOME/bin/pmemd.cuda -O -i mdin -o mdout -p prmtop \
-c inpcrd -r restrt -x mdcrd
```

In this way it is possible to make use of multiple GPUs in a single node for multiple simultaneous calculations. When running a single calculation across multiple GPUs using pmemd.cuda.MPI it also allows the selection of specific GPUs on specific nodes. For example running a 2 GPU job with mpirun -np 2 pmemd.cuda.MPI with the above listed CUDA_VISIBLE_DEVICES would automatically use the second (id=1) and fourth (id=3) GPU in the node. The multi GPU code will avoid assigning MPI GPU tasks to the same GPU if sufficient GPUs are visible. For a more indepth explanation of running GPU accelerated calculations, including how to utilize the peer to peer support in parallel please refer to the GPU section of the AMBER website (http://ambermd.org/gpus/).

18.6.7. Considerations for Maximizing GPU Performance

There are a number of considerations above and beyond those typically used on a CPU for maximizing the performance achievable for a GPU accelerated PMEMD simulation. The following provides some tips for ensuring good performance.

1. Avoid using small values of NTPR, NTWX, NTWV, NTWE and NTWR. Writing to the output, restart and trajectory files too often can hurt performance even on CPU runs; however, this is more acute for GPU accelerated simulations because there is a substantial cost in copying data to and from the GPU. Performance is maximized when CPU to GPU memory synchronizations are minimized. This is achieved by computing as much as possible on the GPU and only copying back to CPU memory when absolutely necessary. There is an additional overhead in that performance is boosted by only calculating the energies when absolutely necessary, hence setting NTPR or NTWE to low values will result in excessive energy calculations. You should typically not set any of these values to less than 100 (except 0 to disable them) and ideally use values of 1000 or more. >100000 for NTWR is ideal, or even better let it just default fo NSTLIM.
2. Avoid setting ntave $\neq 0$. Turning on the printing of running averages results in the code needing to calculate both energy and forces on every step. This can lead to a performance loss of 8% or more when running on the GPU. This can also affect performance on CPU runs although the difference is not as marked. Similar arguments apply to setting the value of ene_avg_sampling to small values.
3. Avoid using the NPT ensemble (ntb=2) when it is not required; if needed make use of the Monte Carlo barostat (barostat=2). Performance will generally be NVE>NVT>NPT (NVT~NPT for barostat=2).
4. Avoid the use of GBSA in implicit solvent GB simulations unless required. The GBSA term is calculated on the CPU and thus requires a synchronization between GPU and CPU memory on every MD step as opposed to every NTPR or NTWX steps when running without this option.
5. Use the Berendsen Thermostat (ntt=1) or Anderson Thermostat (ntt=2) instead of the Langevin Thermostat (ntt=3). Langevin simulations require very large numbers of random numbers which slows performance slightly.

6. Do not assume that for small systems the GPU will always be faster. Typically for GB simulations of less than 150 atoms and PME simulations of less than 5,000 atoms it is not uncommon for the CPU version of the code to outperform the GPU version on a single node. Typically the performance differential between GPU and CPU runs will increase as atom count increases. Additionally the larger the non-bond cutoff used the better the GPU to CPU performance gain will be.

7. When running in parallel across multiple GPUs you should restrict jobs to a single node and select GPUs that are on the same PCIe domain and thus can communicate via peer to peer. For a discussion of PCIe topologies in modern hardware see the following writeup http://tinyurl.com/h469f73. For most budget 4 GPU nodes gpus 0 and 1 can typically communicate via peer to peer and gpus 2 and 3 can communicate via peer to peer. Thus you should not attempt to run a simulation across GPU combinations 0 and 2, or 0 and 3 or 1 and 2 or 1 and 3. The mdout file contains a section that indicates if peer to peer support is enabled.

8. Turn off ECC (Tesla models C2050 and later). ECC can cost you up to 10% in performance. You should verify that your GPUs are working correctly, and not giving ECC errors for example before attempting this. You can turn this off on Fermi based cards and later by running the following command for each GPU ID as root, followed by a reboot:

   `nvidia-smi -g 0 --ecc-config=0` (repeat with -g x for each GPU ID)

   Extensive testing of AMBER on a wide range of hardware has established that ECC has little to no benefit on the reliability of AMBER simulations. This is part of the reason it is acceptable (see recommended hardware) to use the GeForce gaming cards for AMBER simulations. For more details of ECC and MD simulations see the following paper [381].

9. If you see that performance when running multiple - multi-GPU runs is bad. That is that say you run 2 x 2GPU jobs and they don't both run at full speed as if the other job was never running then make sure you turn off thread affinity within your MPI implementation or at least set each MPI thread to use a difference core. In our experience MPICH does not have this on by default and so no special settings are needed however both MVAPICH and OpenMPI set thread affinity by default. This would actually be useful if they did it in an intelligent way. However, it seems they pay no attention to load or even other MVAPICH or OpenMPI runs and always just assign from core 0. So 2 x 2 GPU jobs are, rather foolishly, assigned to cores 0 and 1 in both cases. The simplest solution here is to just disable thread affinity as follows:

   a) MVAPICH: export MV2_ENABLE_AFFINITY=0; mpirun -np 2 ...
   b) OpenMPI: mpirun –bind-to none -np 2 ...

10. Turn on boost clocks if supported. Newer GPUs from NVIDIA, such as the K40 and K80, support boost clocks which allow the clock speed to be increased if there is power and temperature headroom. This must be turned on to obtain optimum performance with AMBER. If you have a K40 or newer GPU supporting boost clocks then run the following:

- `K40: sudo nvidia-smi -i 0 -ac 3004,875` which puts device 0 into the highest boost state.
- `K80: sudo nvidia-smi -i 0 -ac 2505,875`

  To return to normal do: `sudo nvidia-smi -rac`

  To enable this setting without being root do: `nvidia-smi -acp 0`

18.7. Intel® Many Integrated Core Architecture

Note this section refers to the Intel Knights Corner Xeon Phi support that was included in Amber v16 at the time of release. An extensive update is planned for release in late June 2016 to support the Intel Knights Landing Xeon Phi which will be released at that time. You should refer to the Xeon Phi section of the Amber website for the most up to date information (ambermd.org/xeon_phi/).

One of the newest features of PMEMD is support for Intel MIC Products, such as Intel® Xeon Phi^{TM} coprocessors. Intel MIC Architecture support in PMEMD includes both native and offload mode in Amber 16. These versions of the PMEMD engine are considered experimental and there is no guarantee of any performance improvement. However, this support should produce results directly comparable to the CPU implementation due to the floating-point consistency of Intel's processors. Any differences in tests will be a consequence of floating-point rounding differences in hardware. Support for Intel® MIC Architecture in PMEMD is an ongoing project and therefore frequent updates are likely. Improved performance can be expected in upcoming patches.

Support for the Intel MIC Architecture in Amber 1 offers two modes:

- **MIC native mode:** allows all the functionality of the PMEMD engine to be run on Intel® Xeon Phi^{TM} coprocessors in native mode in either serial or parallel (MPI).

- **MIC offload mode:** offloads a portion of the direct sum calculation in a standard PME simulation to the Intel® Xeon Phi^{TM} coprocessor for improved performance. We are anticipating in later releases to extend support to most functionality of PMEMD, which would include offloading to the Intel® Xeon Phi^{TM} coprocessor for Thermodynamic Integration (TI) and Isotropic Periodic Sum (IPS) simulations. Recommended system size to be explored using the MIC offload version of PMEMD is >200,000 atoms in order to benefit from performance improvements. The MIC offload version of PMEMD requires Intel®'s MPI library which is included in the Intel® Cluster Studio XE compiler package.

The supported Intel® Xeon Phi^{TM} product family includes the Intel® Xeon Phi^{TM} coprocessor 3100/5100/7100 series. (*Intel, Xeon, and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.*)

Before attempting to build these versions you should have built and tested the serial and parallel CPU versions of Amber (*pmemd* and *pmemd.MPI*) with the Intel compiler suite and Intel MPI. This will help to ensure that basic issues relating to standard compilation on your hardware and operating system do not lead to confusion with coprocessor related compilation and testing problems. It is recommended that you are familiar with building and running simple code in native/offload mode on an Intel® Xeon Phi^{TM} Coprocessor, which is described in *https://software.intel.com/mic-developer*. You should also be familiar with Amber's compilation procedures.

18.7.1. MIC Native Model

The MIC native version supporting Intel® Xeon Phi^{TM} coprocessors is called *pmemd.mic_native* (or *pmemd.mic_native.MPI* for running simulations in parallel on the coprocessor using MPI) and must be built separately from the standard serial and parallel installations.

Building PMEMD serial for Intel® Xeon Phi^{TM} Coprocessors (native mode)

Assuming you have installed Intel® Parallel Studio XE version 2013 or later, you can build *pmemd.mic_native* as follows:

```
cd $AMBERHOME
./configure -mic_native intel
make install
```

Building PMEMD parallel for Intel® Xeon Phi^{TM} Coprocessors (native mode)

PMEMD parallel for Intel® Xeon Phi^{TM} coprocessors can only be built using the MPI (mpiicc/mpiifort) from the Intel® Cluster Studio XE product, which is supported in Amber through the use of a new MPI flag (-intelmpi). Build *pmemd.mic_native.MPI* as follows:

```
cd $AMBERHOME
./configure -mic_native -intelmpi intel
make install
```

It is possible to run across multiple Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors, even on a cluster, with this implementation. However, it is a functional implementation and not performance optimized at this time. Detailing how to run this way is beyond the scope of the current manual; however to see how to do this with MPI applications in general, see
http://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems .

Running simulations on Intel® Xeon Phi™ Coprocessors (native mode)

In order to run a simulation on the Intel® Xeon Phi™ coprocessor, it is advised that you read the Intel® Xeon Phi™ Coprocessor Developer's Quick Start Guide (*http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide*) in addition to the instructions given here. This guide includes a description of the Intel® Manycore Platform Software Stack (Intel® MPSS), which enables the wide range of usage models that Intel® Xeon Phi™ coprocessors support. Running a simulation on the coprocessor in native mode requires that all files and binaries be visible to the coprocessor. Either mount your file system on the coprocessor (requires root access) or explicitly transfer the binaries, libraries and input files to the */tmp* directory on the coprocessor.

Note: Mounting requires the Amber directory plus any additional libraries used in an Amber simulation to be visible to the coprocessor.

Running simulations with a mounted filesystem:

1. To mount a filesystem please follow instructions available in the Intel MPSS readme file. You can also follow the instructions in *http://software.intel.com/sites/default/files/article/373934/system-administration-for-the-intel-xeon-phi-coprocessor.pdf*
2. Mount your AMBERHOME directory, working directory, Intel compiler directory, Intel MPI_HOME directory (for parallel run), and MKL_HOME directory (if MKL is used) on the coprocessor.
3. Add the following environment variables to a file (source_knc.sh in this example), which will be sourced on execution of *pmemd.mic_native*:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$INTEL_COMPILER_HOME/lib/mic/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MKL_HOME/lib/mic/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MPI_HOME/mic/lib/
export PATH=$PATH:$MPI_HOME/mic/bin
```

4. Run a simulation from your working directory on the host (CPU) with a mounted filesystem:

```
ssh mic0 "source source_knc.sh; \
$AMBERHOME/bin/pmemd.mic_native -O -i mdin -o mdout -p prmtop -c inpcrd"
```

Running simulations without a mounted filesystem:

1. Upload the Intel® Xeon Phi™ coprocessor version of the Intel compiler library to the coprocessor:

```
scp -r $INTEL_COMPILER_HOME/lib/mic/* mic0:/tmp/
```

2. Similarly, upload the coprocessor versions of the MPI and MKL libraries:

```
scp -r $INTEL_MPI_HOME/mic/lib/* mic0:/tmp/
scp -r $INTEL_MPI_HOME/lib/mic/* mic0:/tmp/
```

3. Upload the coprocessor version of PMEMD (*pmemd.mic_native* or *pmemd.mic_native.MPI*) and your working directory (containing the input fies for simulation):

```
scp -r $AMBERHOME/bin/pmemd.mic_native mic0:/tmp/
scp -r working_directory/* mic0:/tmp/
```

4. Change the permissions of the libraries and binaries so that they are executable on the coprocessor:

   ```
   chmod 777 -R /tmp/*
   ```

5. Finally run the simulation from the host:

   ```
   ssh mic0 "export LD_LIBRARY_PATH=/tmp/; cd /tmp; \
   ./pmemd.mic_native -O -i mdin -o mdout -p prmtop -c inpcrd"
   ```

18.7.2. MIC Offload Mode

The MIC offload version supporting Intel® Xeon Phi^{TM} coprocessors is called *pmemd.mic_offload.MPI* and must be built separately from the standard parallel installation. MIC offload is not available in serial.

Building PMEMD for Intel® Xeon Phi^{TM} Coprocessors (offload mode)

Assuming you have installed Intel® Parallel Studio XE version 2013 or later, you can build *pmemd.mic_offload.MPI* as follows:

```
cd $AMBERHOME
./configure -mic_offload intel
make install
```

There is no need to specify the -intelmpi flag as this is the default behavior of the configure script.

Testing PMEMD for Intel® Xeon Phi^{TM} Coprocessors (offload mode)

You can run the test suite using the MIC coprocessor with:

```
make test.mic_offload
```

The majority of these tests should pass. However, given the parallel nature of the MIC coprocessor, meaning the order of operation is not well defined, it is not uncommon for there to be several "possible FAILURES". You should inspect the *.dif* file created in the *$AMBERHOME/logs/test_amber_mic_offload/* directory to manually verify any "possible FAILURES". Differences that occur on only a few lines and are minor in nature can be safely ignored. Any large differences, or if you are unsure, should be posted to the Amber mailing list for comment.

Running simulations on Intel® Xeon Phi^{TM} Coprocessors (offload mode)

Unlike MIC native mode, once PMEMD has been configured with the -mic_offload flag and compiled, no additional steps are required to run *pmemd.mic_offload.MPI*. Work is automatically offloaded to the Intel MIC Architecture.

Execute the following command on the host to run a simulation in MIC offload mode:

```
mpirun -np 8  $AMBERHOME/bin/pmemd.mic_offload.MPI -O
```

Note: Choose the number of MPI processes to suit the specifications of the host CPU, e.g. 8 MPI processes for an Intel Xeon E5-2680 8 core processor, in order to achieve optimum performance. In the initial support for MIC offload in PMEMD, the amount of offloaded work to the coprocessor increases and settles to a stable value after running multiple time steps governed by the Amber load balancer. Thus, it is recommended that the simulation should run for at least 200 time steps to benefit from the coprocessor.

Advanced execution command

The MIC offload code uses OpenMP (OMP) threads to distribute the offloaded work across the MIC coprocessor cores. The default number of OMP threads per offloading MPI process is set to 30; however, this can be overridden by substituting the above execution command with the following advanced execution command in this example runscript:

```
Run.mic_offload
#!/bin/bash
export MIC_ENV_PREFIX=PHI
export OMP_NUM_THREADS=1
mpirun -n 11 ./pmemd.mic_offload.MPI -O \
: -n 1 -env PHI_KMP_PLACE_THREADS 30c,4t,0O \
-env PHI_KMP_AFFINITY scatter \
-env PHI_OMP_NUM_THREADS 30 \
-env MIC_OMP_STACKSIZE 4M \
./pmemd.mic_offload.MPI -O \
: -n 1 -env PHI_KMP_PLACE_THREADS 30c,4t,30O \
-env PHI_KMP_AFFINITY scatter \
-env PHI_OMP_NUM_THREADS 30 \
-env MIC_OMP_STACKSIZE 4M \
./pmemd.mic_offload.MPI -O \
: -n 11 ./pmemd.mic_offload.MPI -O
```

"MIC_ENV_PREFIX=PHI" states that any environment variable prefixed with PHI will be applicable to the MIC coprocessor environment only (not the host processor environment).

"OMP_NUM_THREADS=1" states that the number of host OMP threads is 1.

In the MIC offload version of PMEMD only the middle two MPI processes are responsible for offloading work to the MIC coprocessor, e.g. if 8 MPI processes are specified, threads 4 and 5 are responsible for offloading to the MIC coprocessor. These two MPI processes simultaneously spawn OMP threads on the MIC coprocessor to execute the offloaded chunks of work. By partitioning the execution command to reflect the decomposition strategy, the number of OMP threads can be manually set. Partitioning of an MPI execution command is done via the use of ":" which is demonstrated in the example runscript above.

In this example runscript for a 24 core Intel CPU augmented with a 61 core MIC coprocessor, 24 MPI processes are requested on a single node. The first 11 MPI processes and the last 11 MPI processes execute on the host CPU cores. The middle two MPIs (12 and 13) offload to a single MIC coprocessor and each spawn 30 OMP threads. The cores of the MIC coprocessor are divided in two so that each MPI process is assigned half the cores of the MIC coprocessor. In the above example, MPI process 12 spawns OMP threads on cores 1-30 whereas MPI process 13 spawns OMP threads on cores 31-60.

- "-env PHI_KMP_PLACE_THREADS 30c,4t,0O" states that the MPI process will use 30 cores of the MIC coprocessor (30C), may use as many as 4 threads per core (4T), and the first core that is being used has an offset of 0 (0O). Please consult *https://software.intel.com/en-us/articles/openmp-thread-affinity-control* for a more detailed explanation.
- "–env PHI_KMP_AFFINITY scatter" states that the OpenMP threads will be mapped to the hardware threads in a scattered fashion. Please consult *https://software.intel.com/sites/products/documentation/doclib/stdxe/2013/composerxe/compiler/cpp-lin/index.htm* for other options.
- "-env PHI_OMP_NUM_THREADS 30" states that the MPI process uses 30 OpenMP threads.

It may be of benefit to performance to adjust the number of OMP threads to be spawned by each offloading MPI process depending on the system being simulated, for example:

```
-env  PHI_OMP_NUM_THREADS 60
```

For larger simulations (>200,000 atoms) more OMP threads, such as 2 threads per core (a maximum of 4 threads per core is permitted on the MIC coprocessor), spawned on a MIC coprocessor provides better performance. But for more OMP threads we need a larger stack size per thread and a larger total stack size on a MIC coprocessor. The default stack size is 8 KB.

- "-env MIC_OMP_STACKSIZE 4M" increases the OMP thread stack size to 4 MB.
- For MPSS version 3.2.3 and later the total stack size of a MIC coprocessor is increased by the following steps:

1. On the host, as root, create the directories *etc/* and *etc/security* in */var/mpss/common*:

   ```
   cd /vars/mpss/common
   su
   mkdir -p etc/security
   ```

2. Next create the file *limits.conf* (in */var/mpss/common/etc/security*) containing the following line of text (with tab separated values):

   ```
   "*     soft    stack            unlimited"
   ```

3. Create the file */var/mpss/common.filelist* containing the following (with space separated values) lines:

   ```
   dir /etc/security 755 0 0
   file /etc/security/limits.conf etc/security/limits.conf 644 0 0
   ```

4. Finally, cycle the MPSS daemon to reboot the cards using:

   ```
   service mpss restart
   ```

19. Atom and Residue Selections

There are three ways to select atoms and residues in AMBER-related routines: the **AMBER "mask"** notation, used by most programs, the **NAB "atom expressions"**, which work only with NAB-compiled applications, and an older “GROUP” specification used in *sander* and *pmemd*. Information about these is collected in this chapter.

19.1. Amber Masks

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

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

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

Residue Number List Examples

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

Residue Name List Examples

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

Atom Number List Examples

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

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

Atom Name List Examples

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

Atom Type List Examples

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

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

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

Logical Combinations

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

19.1.1. ambmask

NAME

ambmask - test group input FIND mask (or mask string given in the &cntrl section) and dump the resulting atom selection in a given format

SYNOPSIS

```
ambmask -p prmtop -c inpcrd -prnlev [0-3] -out [short| pdb| amber] -find [maskstr]
```

DESCRIPTION

ambmask acts as a filter which takes Amber topology and coordinate "restart" file and applies the "maskstr" selection string (similar syntactically to UCSF Chimera/Midas) to select specific atoms or residues. Residues can be selected by their numbers or names. Atoms can be selected by numbers, names, or Amber (forcefield) type. Selections are case insensitive. The selected atoms are printed to **stdout** (by default, in Amber-style PDB format). Atom and residue names and numbers are taken from Amber topology. Beware that selection string works on those names and not the ones from the original PDB file. If you are not sure how atoms or residues are named or numbered in the Amber topology, use **ambmask** with a selection string ":*" (which is the default) to dump the whole PDB file with corresponding Amber atom/residue names and numbers.

The "maskstr" selection expression is composed of "elementary selections". These start with ":" to select by residues, or "@" to select by atoms. Residues can be selected by numbers (given as numbers separated by commas, or as ranges separated by a dash) or by names (given as a list of residue names separated by commas). The same holds true for atom selections by atom numbers or atom names. In addition, atoms can be selected by Amber atom type, in which case "@" must be immediately followed by "%". ":*" means all residues and "@*" means all atoms. The following examples show the usage of this syntax. Square brackets should not be used in actual expressions, they are only used for clarity here:

```
:{residue numlist} [:1-10] [:1,3,5] [:1-3,5,7-9]
:{residue namelist} [:LYS] [:ARG,ALA,GLY]
@{atom numlist} [@12,17] [@54-85] [@12,54-85,90]
@{atom namelist} [@CA] [@CA,C,O,N,H]
@%{atom typelist} [@%CT] [@%N*,N3]
```

These "elementary selections" can be combined into more complex selections using binary operators "&" (and) and "|" (or), unary operator "!" (negation), distance binary operators "<:", ">:", "<@", ">@", and parentheses. Spaces around operators are irrelevant. Parentheses have the highest priority, followed by distance operators ("<:", ">:", "<@", ">@"), "!" (negation), "&" (and) and "|" (or) in order of descending priority. A wildcard "=" in an atom or residue name matches any name starting with a given character (or characters). For example, [:AS=] would match all aspartic acid residues (ASP), and asparagines (ASN); [@H=] would match all atom names starting with H (which are effectively all hydrogens). It cannot be used to match the end part of names (such as [:=A]). Some examples of more complex selections follow:

```
[@C= & !@CA,C]
```

.. all carbons except backbone alpha and carbonyl carbon

```
[(:1-3@CA | :5-7@CB)]
```

.. alpha carbons in residues 1-3 and beta carbons in residues 5-7

```
[:CYS,ARG & !(:1-10 | @CA,CB)]
```

.. all CYS and ARG atoms except those which are in residues 1-10 and which are CA or CB

```
[:* & !@H=] or [!@H=]
```

.. all heavy atoms (i.e. except hydrogens)

```
[:5 <@4.5]
```

.. all atoms within 4.5A from residue 5

```
[(:1-55 <:3.0) & :WAT]
```

.. all water molecules within 3A from residues 1-55

Compound expressions of the following type are also allowed:

```
:{residue numlist|namelist}@{atom numlist|namelist|typelist}
[:1-10@CA] is equivalent to [:1-10 & @CA]
[:LYS@H=] is equivalent to [:LYS & @H=]
```

OPTIONS

The program needs an Amber topology file and coordinates (restrt format). The filename specified with the *-p* option is Amber topology, while the filename given with the *-c* option is a coordinate file. If *-p* or *-c* options are not given, the program expects that files "prmtop" and/or "inpcrd" exist in the current directory, which will be taken as topology and coordinate files correspondingly. If no command line options are given, the program prints the usage statement.

The option *-prnlev* specifies how much (debugging) information is printed to **stdout**. If it is 0, only selected atoms are printed. More verbose output (which might be useful for debugging purposes) is achieved with higher values: 1 prints original "maskstr" in its tokenized (with operands enclosed in square brackets) and postfix (or Reverse Polish Notation) forms; number of atoms and residues in the topology file and number of selected atoms are also printed to **stdout**. 2 prints the resulting mask array, which is an array of integer values, with '1' representing a selected atom, and '0' an unselected one. Value of 3, in addition, prints mask arrays as they are pushed or popped from the stack (this is really only useful for tracing the problems occurring during stack operations). The *-prnlev* values of 0 or 1 should suffice for most uses.

The option *-out* specifies the format of printed atoms. "short" means a condensed output using residue (:) and atom (@) designators followed by residue ranges and atom names. "pdb" (default) prints atoms in Amber-style PDB format with the original "maskstr" printed as a REMARK at the top of the PDB file, and "amber" prints atom/residue ranges in the format suitable for copying into group input section of Amber input file.

The option *-find* is followed by "maskstr" expression. This is a string where some characters have a special meaning and thus express what parts (atoms/residues) of the molecule will get selected. The syntax of this string is explained in the section above (DESCRIPTION). If this option is left out, it defaults to ":*", which selects all atoms in the given topology file. The length of "maskstr" is limited to 80 characters. If the "maskstr" contains spaces or special characters (which would be expanded by the shell), it should be protected by single or double quotes (depending on the shell). In addition, for C-shells even a quoted exclamation character may be expanded for history substitution. Thus, it is recommended that the operand of the negation operator always be enclosed in parentheses so that "!" is always followed by a "(" to produce "!(" which disables the special history interpretation. For example, [@C= & !(@CA,C)] selects all carbons except backbone alpha and carbonyl carbon; the parentheses are redundant but shell safe. The man page indicates further ways to disable history substitution.

FILES

Assumes that *prmtop* and *inpcrd* files exists in the current directory if they are not specified with -*p* and -*c* options. Resulting (i.e. selected) atoms are written to **stdout**.

BUGS

Because all atom names are left justified in Amber topology and the selections are case insensitive, there is no way to distinguish some atom names: alpha carbon CA and a calcium ion Ca are a notorious example of that.

19.2. "Atom Expressions" in NAB Applications

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

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

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

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

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

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

19.3. GROUP Specification

This section describes the format used to define groups of atoms in various Amber programs. In *sander*, a group can be specified as a movable "belly" while the other atoms are fixed absolutely in space (aside from scaling caused by constant pressure simulation), and/or a group of movable atoms can independently restrained (held by a potential) at their positions. In *anal*, groups can be defined for energy analysis.

Except in the analysis module where different groups of atoms are considered with different group numbers for energy decomposition, in all other places the groups of atoms defined are considered as marked atoms to be included for certain types of calculations. In the case of constrained minimization or dynamics, the atoms to be constrained are read as groups with a different weight for each group.

Reading of groups is performed by the routine RGROUP, and you are advised to consult it if there is still some ambiguity in the documentation.

Input description:

```
- 1 - Title format(20a4)
ITITL Group title for identification.
Setting ITITL = 'END' ends group input.
--------------------------------------------------------------------------
- 1A - Weight format(f)
This line is only provided/read when using GROUP input to
define restrained atoms.
WT The harmonic force constants in kcal/mol-A**2 for the group
of atoms for restraining to a reference position.
--------------------------------------------------------------------------
- 1B - Control to define the group
KTYPG , (IGRP(I) , JGRP(I) , I = 1,7) format(a,14i)
KTYPG Type of atom selection performed. A molecule can be
defined by using only 'ATOM' or 'RES', or part of the
molecule can be defined by 'ATOM' and part by 'RES'.
'ATOM' The group is defined in terms of atom numbers. The atom
number list is given in igrp and jgrp.
'RES' The group is defined in terms of residue numbers. The
residue number list is given in igrp and jgrp.
'FIND' This control is used to make additional conditions
(apart from the 'ATOM' and 'RES' controls) which a given
atom must satisfy to be included in the current group.
The conditions are read in the next section (1C) and are
terminated by a SEARCH card.
Note that the conditions defined by FIND filter any set(s) of atoms
defined by the following ATOM/RES instructions. For example,
-- group input: select main chain atoms --
FIND
* * M *
SEARCH
RES 1 999
END
END
'END' End input for the current group. Followed by either another
group definition (starting again with line 1 above), or by a second
'END' "card", which terminates all group input.
IGRP(I) , JGRP(I)
The atom or residue pointers. If ktypg .eq. 'ATOM' all
atoms numbered from igrp(i) to jgrp(i) will be put into
the current group. If ktypg .eq. 'RES' all atoms in the
residues numbered from igrp(i) to jgrp(i) will be put
into the current group. If igrp(i) = 0 the next control
card is read.
It is not necessary to fill groups according to the
numerical order of the residues. In other words, Group 1
could contain residues 40-95 of a protein, Group 2 could
contain residues 1-40 and Group 3 could contain residues
96-105.
If ktypg .eq. 'RES', then associating a minus sign with
```

```
igrp(i) will cause all residues igrp(i) through jgrp(i)
to be placed in separate groups.
In the analysis modules, all atoms not explicitly defined
as members of a group will be combined as a unit in the
(n + 1) group, where the (n) group in the last defined
group.
------------------------------------------------------------------------
- 1C - Section to read atom characteristics
***** Read only if KTYPG = 'FIND' *****
JGRAPH(I) , JSYMBL(I) , JTREE(I) , JRESNM(I) format(4a)
A series of filter specifications are read. Each filter consists
of four fields (JGRAPH,JSYMBL,JTREE,JRESNM), and each filter is placed
on a separate line. Filter specification is terminated by a line with
JGRAPH = 'SEARCH'. A maximum of 10 filters may be specified for a
single 'FIND' command.
The union of the filter specifications is applied to the atoms defined
by the following ATOM/RES cards. I.e. if an atom satisfies any of the
filters, it will be included in the current group. Otherwise, it is not
included. For example, to select all non main chain atoms from residues
1 through 999:
-- group input: select non main chain atoms --
FIND
* * S *
* * B *
* * 3 *
* * E *
SEARCH
RES 1 999
END
END
'END' End input for the current group. Followed by either another
The four fields for each filter line are:
JGRAPH(I) The atom name of atom to be included. If this and the
following three characteristics are satisfied the atom is
included in the group. The wild card '*' may be used to
to indicate that any atom name will satisfy the search.
JSYMBL(I) Amber atom type of atom to be included. The wild card
'*' may be used to indicate that any atom type will
satisfy the search.
JTREE(I) The tree name (M, S, B, 3, E) of the atom to be included.
The wild card '*' may be used to indicate that any tree
name will satisfy the search.
JRESNM(I) The residue name to which the atom has to belong to be
included in the group. The wild card '*' may be used to
indicate that any residue name will satisfy the search.
------------------------------------------------------------------------
```

Examples:

The molecule 18-crown-6 will be used to illustrate the group options. This molecule is composed of six repeating (-CH2-O-CH2-) units. Let us suppose that one created three residues in the PREP unit: CRA, CRB, CRC. Each of these is a (-CH2-O-CH2-) moiety and they differ by their dihedral angles. In order to construct 18-crown-6, the residues CRA, CRB, CRC, CRB, CRC, CRB are linked together during the LINK module with the ring closure being between CRA(residue 1) and CRB(residue 6).

Input 1:

```
Title one
RES 1 5
END
```

```
Title two
RES 6
END
END
```

Output 1: Group 1 will contain residues 1 through 5 (CRA, CRB, CRC, CRB, CRC) and Group 2 will contain residue 6 (CRB).

Input 2:

```
Title one
RES 1 5
END
Title two
ATOM 36 42
END
END
```

Output 2: Group 1 will contain residues 1 through 5 (CRA, CRB, CRC, CRB, CRC) and Group 2 will contain atoms 36 through 42. Coincidentally, atoms 36 through 42 are also all the atoms in residue 6.

Input 3:

```
Title one
RES -1 6
END
END
```

Output 3: Six groups will be created; Group 1: CRA, Group 2: CRB,..., Group 6: CRB.

Input 4:

```
Title one
FIND
O2 OS M CRA
SEARCH
RES 1 6
END
END
```

Output 4: Group 1 will contain those atoms with the atom name ’O2’, atom type ’OS’, tree name ’M’ and residue name ’CRA’.

Input 5:

```
Title one
FIND
O2 OS * *
SEARCH
RES 1 6
END
END
```

Output 5: Group 1 will contain those atoms with the atom name ’O2’, atom type ’OS’, any tree name and any residue name.

Input 6:

```
Title one
RES 1 3 6 6
END
END
```

Output 6: One group is created containing residues 1 to 3 and 6. Up to seven ranges of contiguous residues can be specified per group. (In this case there are two ranges).

20. Sampling configuration space

There are many instances when standard molecular dynamics simulations get "stuck" near the starting configuration, and fail to adequately sample the available low-energy configurational space. This chapter describes a variety of techniques that can partially overcome such problems. The following chapter (on Free Energies) continues many of these ideas, adapting them to the calculation of alchemical or configurational free energy differences. There is no good distinction between these two chapters, because good sampling of the canonical distribution and estimation of free energies go hand-in-hand. But the present chapter covers methods that are *primarily* devoted to enhanced or accelerated sampling, whereas the following chapter considers methods that explicitly estimate free energy differences.

20.1. Self-Guided Langevin dynamics

Self-guided Langevin dynamics (SGLD) is designed to enhance conformational search efficiency in either a molecular dynamics (MD) simulation (when *gamma_ln=0*) or a Langevin dynamics (LD) simulation (when *gamma_ln>0*). This method accelerates low frequency motion to enhance conformational sampling. [382–385]

Overview: The input parameter, *tsgavg*, defines the lower limit period of the low frequency motion. Typically, *tsgavg*=0.2 ps is recommended for motions like phase separation, secondary structure folding, and ligand docking, while *tsgavg*=1.0 ps is recommended for protein domain motion, and protein-protein docking. The input parameter, *sgft* or *tempsg*, defines the strength of the guiding effect. *sgft*=0~1 with 0 for regular LD or MD simulations. A smaller *sgft* will produce results closer to a normal MD or LD simulation. *tempsg* defines a conformational search ability that is comparable to a high temperature simulation at *temp*0=*tempsg*. Normally, *tempsg* or *sgft* is set to accelerates slow events to an affordable time scale while minimizing the pertubation to the conformational distribution. The guiding force can be applied to a part of a simulation system between atom *isgsta* and atom *isgend*.

The conformational distribution of SGLD can be reweighted to produce canonical ensemble averages[383, 386]. The reweighting information is in the simulation output file. The force-momentum based SGLD algorithm (SGLDfp) [387] (*isgld*=2) is available to allow conformational search to be accelerated while the canonical ensemble distribution is mantained. However, the conformational searching abilities of SGLDfp is reduced as compared with SGLD (*isgld*=1). Most recently, the generalized SGLD method [385](SGLDg) (*isgld*=3) is developed to enhance conformational search in both LD (when *gamma_ln>0*) and MD (when *gamma_ln=0*) simulations. SGLDg is more convenient and flexible and has better characterized conformational distribution and can be an replacement of normal LD to sample the canonical ensemble by setting *tempsg*=*temp*0.

- SGLDfp (*isgld*=2) allows low resolution structures, such as secondary structures and tertiary structures, and/or high resolution structures, such as bond lengths and bond angles, to be canonically sampled.
- SGLDg (*isgld*=3) utilizes independent low-frequency and high frequency Langevin equations to characterize the conformational searching and distribution. It provides options to use the force guiding factor, *sgff*, to control energy barriers in low frequency space and to use the guiding temperature, *tempsg*, and the momentum guiding factor, *sgft*, to enhance low frequency motion. When *tempsg* is set to the simulation temperature (*tempsg*=*temp*0), SGLDg samples exactly the canonical ensemble and is an excellent replacement of regular Langevin dynamics with enhanced conformational sampling ability, especially when the friction constant is high, for example, *gamma_ln*>10/ps. SGLDg is recommanded when cannonical distribution need be maintained(*tempsg*=*temp*0) or reweighted(*tempsg*>*temp*0).

SGLD can be used for replica exchange simulations (RXSGLD)[388] to achieve enhanced sampling with or without elevating temperature. *sgft* or *tempsg* can be used to define different replicas. See Section 21.5.5 for a detailed description of RXSGLD.

isgld	SGLD algorithm index. Default *isgld*= 0, SGLD is diabled; *isgld*=1 will turn on SGMD/SGLD method for accelerated conformational search; *isgld*=2 will turn on the SGLDfp method [387] to mantain a canonical ensemble distribution. *isgld*=3 will enable SGLDg/SGMDg method.
tsgavg	Local averaging time (*psec*) for the guiding force calculation. Default 0.2 *psec*. A larger value defines slower motion to be enhanced.
sgft	Momentum guiding factor. Defines the strength of the guiding effect. Default 1.0 when *gamma_ln>0* (SGLD), 0.2 when *gamma_ln=0* (SGMD), or 0 for SGLDg or 1 for SGMDg (*isgld*=3). *When isgld=1 or 2, tempsg*>0 will override *sgft*. When *isgld*=3 and *gamma_ln*=0, i.e., SGMDg, *sgft* represents the guiding friction constant in the unit of 1/ps and should be set to a positive value.
sgff	Force guiding factor for SGLDg or SGMDg (*isgld*=3). *sgff* is used to scale down low frequency energy surface by a factor, (1+*sgff*). *sgff* is suggested to take values between 0 and -0.1, with default value of 0. *sgff* is effective for SGMDg (*isgld*=3 and *gamma_ln*=0) simulations .
tempsg	Target guiding temperature (*K*). This parameter is redefined since Amber 12 as a conformational search ability which is comparable to a high temperature simulation with *temp0=tempsg*. For example, by seting *tempsg*=500K, a SGMD/SGLD simulation will accelerate conformational search as much as rising the simulation temperature to 500K. When *isgld=1* or *2*, the default is *tempsg=0* K. When *tempsg=0*, the guiding effect will be defined by *sgft*. *tempsg*>*temp*0 will accelerate a conformational search and *tempsg*<*temp*0 will slow down a conformational search. *When isgld=1 or 2,* once *tempsg* is set, *sgft* will fluctuate to reach the target conformational search ability. When *isgld=3* (SGLDg), the default is *tempsg*=*temp*0, which allows the canonical ensemble will be sampled. When *tempsg<>temp0,* the reweighting factor can be found in the SGLD output lines described below.
isgsta	The first atom index of SGLD region. Default is 1.
isgend	The last atom index of SGLD region. Default is *natom*.
fixcom	Option to remove the net translation of the center of mass. For finite systems it is often more convenient to have the center of mass fixed. Default 0 *when gamma_ln>0 or 1 when gamma_ln=0.* When *fixcom>0,* the center of mass is fixed.
treflf	Reference low frequency temperature. Default 0.0. *treflf* is the low frequency temperature when no guiding force is applied to a simulation system (*sgft*=0). *treflf* is required for the weighting factor calculation in SGMD/SGLD simulation (*isgld*=1) or for the guiding force calculation in SGMDfp/SGLDfp simulations (*isgld*=2). *treflf* is not needed in SGLDg or SGMDg (*isgld*=3) simulations. When *treflf*=0, *treflf* will be estimated during a simulation. An accurate value of *treflf* can increase the accuracy in the weighting factor calculation in SGMDfp/SGLDfp simulations.
sgfd	Optional high frequency force guiding factor for SGLDfp (*isgld*=2). Should not be set if conformational distribution in high resolution need be maintained. When not set, its instantaneous value (printed out in simulation output files) fluctuates to maintain high resolution conformational distribution.

The output of SGMD/SGLD simulations contains the following properties related to the enhancement in conformational search and reweighting of conformational distribution:

SGLF = SGFT TEMPSG TEMPLF TREFLF FRCLF EPOTLF SGWT
SGHF = SGFF SGFD TEMPHF TREFHF FRCHF EPOTHF VIRSG

These quantities are instantaneous values defined as below:

SGFT: Momentum guiding factor,
SGFF: Force guiding factor. Adjusted in SGLDfp simulations (*isgld*=2)
SGFD: Force dumping factor. Adjusted in SGLDfp simulations (*isgld*=2)
TEMPSG: Guiding temperature.
SGWT: Weighting free energy. exp(SGWT) is the weighting factor of current frame.
VIRSG: Virial of the guiding force.

TEMPLF: low frequency temperature
TEMPHF: high frequency temperature
TREFLF: reference low frequency temperature. It is the TEMPLF at SGFT=0 and TEMPSG=0.
TREFHF: reference high frequency temperature. It is the TEMPHF at SGFT=0 and TEMPSG=0.
FRCLF: low frequency force factor
FRCHF: high frequency force factor
EPOTLF: low frequency potential energy
EPOTHF: high frequency potential energy

The weight of a conformation is calculated by

```
Weight=exp(SGWT) =exp(((FRCLF*TREFLF/TEMPLF-1)*EPOTLF+
(FRCHF*TREFHF/TEMPHF-1)*EPOTHF+VIRSG)/(KBOLTZ*Temp))
```

or:

$$w_i = \exp\left((\lambda_{LF}\frac{T^0_{LF}}{T_{LF}} - 1)\frac{E_{LF}}{kT} + (\lambda_{HF}\frac{T^0_{HF}}{T_{HF}} - 1)\frac{E_{HF}}{kT} + \frac{W_{sg}}{kT} \right)$$

For convenience, two scripts, *sgldinfo.sh* and *sgldwt.sh*, are provided in the AMBERHOME/bin directory to extract SGLD properties and weighting factors from sander output files. For example, one can run:

```
sgldinfo.sh mdout
```

to examine the SGLD properties, and run:

```
sgldwt.sh mdout
```

to print weighting factors at each print time frame. One may specify TREFLF, e.g., 23.5 K, and/or TREFHF, e.g., 278.2 K, for more accurate weighting factors:

```
sgldwt.sh mdout 23.5 278.2
```

TREFLF and TREFHF can be obtained with *sgldinfo.sh* from a SGLD simulation at the same condition except SGFT=0 and TEMPSG=0. Without specifying TREFLF and TREFHF, they will be estimated for the calculation. Ensemble average properties are calculated through reweighting:

$$< P >= \frac{\sum_{i=N_0}^{N} w_i P_i}{\sum_{i=N_0}^{N} w_i}$$

For SGLDfp (*isgld*=2) and SGLDg (*isgld*=3) simulations, no reweighting is needed.

Here is an example of a SGLD simulation input file:

```
Sample SGLD simulation to reach a 500K conformational search ability
&cntrl
nstlim=1000, cut=99.0, igb=1, saltcon=0.1,
ntpr=100, ntwr=100000, ntt=3, gamma_ln=10.0,
ntx=5, irest=1, ig = 256251,
ntc=2, ntf=2, tol=0.000001,
dt=0.002, ntb=0, tempi=300., temp0=300.,
isgld=1,tsgavg=0.2,tempsg=500,
/
```

Below is an example of a SGLDg simulation input file to sample the canonical ensemble while accelerate conformational search:

```
Sample SGLDg simulation for efficient conformational sampling
&cntrl
nstlim=1000, cut=99.0, igb=1, saltcon=0.1,
ntpr=100, ntwr=100000, ntt=3, gamma_ln=10.0,
ntx=5, irest=1, ig = 256251,
ntc=2, ntf=2, tol=0.000001,
dt=0.002, ntb=0, tempi=300., temp0=300.,
isgld=3,tsgavg=0.2,tempsg=300,sgft=0.5,sgff=-0.1,
/
```

Below is an example of a SGMDg simulation input file for accelerated conformational search:

```
Sample SGLDg simulation for efficient conformational sampling
&cntrl
nstlim=1000, cut=99.0, igb=1, saltcon=0.1,
ntpr=100, ntwr=100000, ntt=1, gamma_ln=0.0,
ntx=5, irest=1, ig = 256251,
ntc=2, ntf=2, tol=0.000001,
dt=0.002, ntb=0, tempi=300., temp0=300.,
isgld=3,tsgavg=0.2,tempsg=500,sgft=1,sgff=-0.1,
/
```

20.2. Accelerated Molecular Dynamics

20.2.1. Introduction

Many systems of interest in chemistry, physics and biology are characterized by the presence of a number of metastable states separated by large barriers. Correctly sampling these systems is challenging for methods based on Molecular Dynamics, Monte Carlo sampling or any other type of dynamic simulation. For most biological systems of interest, the simulation time is limited to the nanosecond-microsecnd time scale, so simple molecular dynamics cannot be used to adequately explore portions of the energy landscape separated by high barriers from the initial minimum. Furthermore, for most biological molecules, the energy landscape has multiple minima or potential energy wells with high free energy barriers, and during a molecular dynamics simulation the system is trapped in one or another local minimum for long periods of simulation time. Consequently, thermodynamics and many other properties of interest for large biological systems cannot be simulated directly because of the nonergodic nature of the present state of the molecular dynamics methodology for systems with high free energy barriers.

Accelerated Molecular Dynamics (aMD) is a bias potential introduced by the McCammon group at UCSD [389]. It is a modification to the potential that in practice reduces the height of local barriers, allowing the calculation to evolve much faster. A number of methods have been suggested to aid this problem, like replica exchange, metadynamics, etc. AMD represents an interesting option as it only requires the evolution of a single copy of the system, plus it doesn't require any previous knowledge of the shape of the potential, i.e. aMD doesn't require information of where are the barriers, saddle points or even what type of configuration changes are expected or necessary to traverse through a particular barrier. Moreover, an interesting feature of aMD is that the shape of the added potential conserves the underlying shape of the real one, such that minima are maintained as minima and barriers are preserved as barriers. In result, adding the aMD potential in practice simply modifies the relation between energy differences, so the distribution of sampling of different structures is still related to the original potential distribution and can be recovered exactly by reweighing.

The aMD modification of the potential is defined by the following equation:

$$\mathrm{V(r)}* = V(r) + \Delta V(r) \tag{20.1}$$

$$\Delta V(r) = \frac{(Ep - V(r))^2}{(\alpha P + Ep - V(r))} + \frac{(Ed - Vd(r))^2}{(\alpha D + Ed - Vd(r))} \tag{20.2}$$

where $V(r)$ is the normal potential and $Vd(r)$ is the normal torsion potential. Ep and Ed are average potential and dihedral energies that serve as a reference energy from which to compare the present position of the calculation and therefore the relationship to the boosting factor to be applied. The terms αP and αD are factors that determine inversely the strength with which the boost is applied. For large values of alpha, the potential felt at any point will essentially be the same as the true potential For values of alpha close to zero, the potential felt becomes constant, in this limit, the sampling becomes a random walk. The amount of boost felt at a particular point in the calculation, therefore, depends on the present value of the potential and dihedral energy, which is in direct correlation to how low in the energy surface the configuration is positioned at that moment. The boosting potential will be proportionally bigger for deeper regions of the potential energy surface, while it will be smaller for higher points, which in result conserves the underlying shape of the potential, as previously mentioned.

AMD has been applied to a vast diversity of interesting problems [390–394]. We have recently applied the implementation of aMD in Amber to the Bovine Pancreatic Trypsin Inhibitor and compared with an unbiased millisecond MD simulation, showing aMD is able to recover the right population distribution and shows excellent agreement with the MD simulation as with experimental data [393].

20.2.2. AMD implementation in Amber

AMD has been implemented in both *sander* and *pmemd* by Romelia Salomon-Ferrer. The implementation includes the possibility of boosting independently only the torsional terms of the potential (*iamd*=2) or the whole potential at once (*iamd*=1). It also allows the possibility to boost the whole potential with an extra boost to the torsions(*iamd*=3). All the information generated by aMD, necessary for reweighing is stored at each step into a vector which is flushed to a log file (*amd.log* by default) every time the coordinates are written to disk, i.e. every *ntwx* steps. This is done for performance reasons, since writing to disk is always time consuming and it is not advisable to do it every step. The name of the log file can be set to a user defined name by using the command line option -amdlog when running Amber. Our present implementation also allows the user to delay (or lag) the boosting a number of steps, i.e. only boost with a particular frequency defined by the variable *amdlag*. Additional parameters are specified by the following variables: *EthreshD* (Ed), *alphaD* (αD), *EthreshP* (Ep) and *alphaP* (αP).

AMD output is saved the *amd.log* this file contains all the information needed for reweighting the results obtained to recover the unperturbed distributions. The *amd.log* file gets written with the same frequency at which the configurations are saved to disk in the trajectory file (*mdcrd*). Each line corresponds to the information of a corresponding snapshot being saved on the mdcrd file. Regardless of what iamd value is used, the number of columns in the amd.log file are always the same, they just have 0 or 1 (correspondingly) if no boost is being added to dihedral or total energy

The amd.log file has the following header:

```
#All energy terms stored in units of kcal/mol
#ntwx,total_nstep,Unboosted-Potential-Energy,Unboosted-Dihedral-Energy,Total-Force-Weight,
Dihedral-Force-Weight,Boost-Energy-Potential,Boost-Energy-Dihedral
```

The description for the main columns is as follows:

- **Unboosted-Potential-Energy:** Total Potential Energy without boost added, kcal/mol.
- **Unboosted-Dihedral-Energy:** dihedral energy without boost added, kcal,mol**.**
- **Total-Force-Weight:** The force scaling factor calculated from the boost to the Total Potential Energy
- **Dihedral-Force-Weight:** The dihedral force scaling factor from dihedral boost
- **Boost-Energy-Potential:** The boost energy in kcal/mol
- **Boost-Energy-Dihedral:** The dihedral boost energy in kcal/mol

IMPORTANT NOTE: Before Amber 14 the boost energy for the dihedral and total potential energy (last two columns) was given in units of *kT*. This decision was made at the beginning with the idea that the user could read and use these values directly for reweighting without any further work, but later it we decided it was much better and more consistent to have all energy output in kcal/mol as the rest of AMBER's energy output. As of AMBER 14, the last two columns of the amd.log file as given in units of kcal/mol.

Reweighting

For reweighting aMD results we would like to add the link to a great tutorial (http://mccammon.ucsd.edu/computing/amdReweighting/) which also provides a small python script to perfom the reweighting. The script is compatible with the newer versions of AMBER, and can be used to reweight 1D and 2D distributions. A simple C code is also provided in the aMD tutorial that performs reweighting based on the Kernel Density Estimation algorithm. This algorithm also performs very well and reduces the amount of noise without using a truncated expression for the exponential and can be used as an alternative for reweighting. To extract the energies from the amd.log file into a file, weights.dat, to use with this script, something like the following could be done:

```
# Column 1: dV in units of kbT; column 2: timestep; column 3: dV in units of kcal/mol
# For AMBER14: # awk 'NR%1==0' amd.log | awk '{print ($8+$7)/(0.001987*300)

    " " $2 " " ($8+$7)}' > weights.dat

# For AMBER12: # awk 'NR%1==0' amd.log | awk '{print ($8+$7)" " $3

    " " ($8+$7)*(0.001987*300)}' > weights.dat
```

For reweighting a 2D distribution, for instance a Phi Psi distribution, you would need to extract the values for Phi and Psi for each frame in the mdcrd file using AmberTools and generate the file Phi_Psi file and then use the python tool provided in the website to get the reweighted surface.

```
python PyReweighting-2D.py -input Phi_Psi -Emax 100 -discX 6 -discY 6


    -job amdweight_MC -order 10 -weight weights.dat | tee -a reweight_variable.log
```

For reweighting using a Maclaurin series expansion as an approximation for the exponential weight.

20.2.3. Preparing a system for aMD

As mentioned before, running aMD requires the definition of few parameters. AMD parameters are determined based on previous knowledge of the system, which is easily acquirable by a short regular MD simulation, from which the average values of the potential and torsion energy can be estimated. From there, a given amount of energy per degree of freedom is added those values, in the form of multiples of alpha, setting the values of Ep and Ed to be used. The following example should help clarify this procedure.

```
Average Dihedral : 611.5376 (based on MD simulations)
Average EPtot : -53155.3104 (based on MD simulations)
total ATOMS=16950
protein residues=64
```

For the dihedral potential:

```
Approximate energy contribution per degree of freedom.
3.5*64= 224          The value of 3.5 kcal/mol/residue seems to work well
alphaD = (1/5)*224 = 45 The value of .2 seems to work well
EthreshD = 224+611 = 835
For the total potential
alphaP = 16950*(1/5)=3390
For a lower boost you can also use a value between 0.15-0.19 instead of 0.20 (0.16 works well)
EthreshP = -53155.3104 + 3390 = -49765.3104
With these parameters, the aMD parameters in the input file should then be set to
```

```
iamd=3,EthreshD=835,alphaD=45,EthreshP=-49765,alphaP=3390,
For a higher acceleration it is common to simply add to Eb(dih) multiples of alpha. In this example
iamd=3,EthreshD=880,alphaD=45,EthreshP=-49765,alphaP=3390,
Two levels higher would be then defined by:
iamd=3,EthreshD=925,alphaD=45,EthreshP=-49765,
```

After the aMD parameters to be used are defined, an MD run with aMD can be set using those parameters. Depending on the progress of the simulation, a higher boost can be applied as specified in the above example.

20.2.4. Sample input file for aMD

An example of an input file would be the following:

```
AVP dt=2.0fs with SHAKE, NPT aMD boost pot and dih
 &cntrl
  imin=0, irest=1, ntx=5,
  dt=0.002, ntc=2, ntf=2, tol=0.000001,iwrap=1,
  ntb=2, cut=12.0, ntp=1,igb=0,ntwprt = 3381,ioutfm = 1,
  ntt=3, temp0=310.0, gamma_ln=1.0, ig=-1,
  ntpr=1000, ntwx=1000, ntwr=2000000, nstlim=2000000,
  iamd=3,EthreshD=835,
  alphaD=45,EthreshP=-49765,
  alphaP=3390,
 /
&ewald
dsum_tol=0.000001,
 /
```

20.2.5. Further information

Test cases have been included into the distribution of Amber, also a tutorial based on a study we performed on BPTI [393], showing the power of aMD and its validation versus a millisecond run on the same system performed on Anton is now present on the Amber website. We encourage the user to read the paper, as well as follow the tutorial for more information.

20.3. Gaussian Accelerated Molecular Dynamics

Gaussian Accelerated Molecular Dynamics (GaMD) is a biomolecular enhanced sampling method that works by adding a harmonic boost potential to smoothen the system potential energy surface. The boost potential follows Gaussian distribution, which allows for accurate reweighting using cumulant expansion to the second order. GaMD has been demonstrated on simulations of alanine dipeptide, chignolin folding and ligand binding to the T4-lysozyme [395]. GaMD enables unconstrained enhanced sampling of these biomolecules without the need to set predefined reaction coordinates. Furthermore, the free energy profiles obtained from reweighting of the GaMD simulations help identify distinct low energy states of the biomolecules and characterize the protein folding and ligand binding pathways quantitatively.

The basic theory of GaMD can be found in References [395, 396], or at http://gamd.ucsd.edu.

GaMD has been implemented in pmemd, both the serial and parallel versions on CPU (*pmemd* and *pmemd.MPI*) and GPU (*pmemd.cuda* and *pmemd.cuda.MPI*) by Yinglong Miao. Note that GaMD is not available in Sander. Similar to aMD, GaMD provides options to add only the total potential boost (*igamd*=1), only dihedral potential boost (*igamd*=2), or the dual potential boost (igamd=3). The dual-boost simulation generally provides higher acceleration than the other two types of simulations for enhanced sampling. The simulation parameters comprise of the threshold energy values and the effective harmonic force constants, and for the total and dihedral potential boost, respectively. All the information generated by GaMD, necessary for reweighing is stored at each step into a vector which is flushed to a log file (*gamd.log* by default) every time the coordinates are written to disk, i.e. every *ntwx* steps. The name of the log file can be set to a user defined name by using the command line option *-gamdlog* when running Amber. Additional parameters are specified by the following variables:

igamd	Flag to apply boost potential **= 0** (default) no boost is applied **= 1** boost on the total potential energy only **= 2** boost on the dihedral energy only **= 3** dual boost on both dihedral and total potential energy
iE	Flag to set the threshold energy E **= 1** (default) set the threshold energy to the lower bound E = Vmax **= 2** set the threshold energy to the upper bound $E = V_{min} + (V_{max} - V_{min})/k_0$
ntcmdprep	The number of preparation conventional molecular dynamics steps. This is used for system equilibration and the potential energies are not collected for calculating their statistics. The default is 200,000 for a simulation with 2 fs timestep.
ntcmd	The number of initial conventional molecular dynamics simulation steps. Potential energies are collected between *ntcmdprep* and *ntcmd* to calculate their maximum, minimum, average and standard deviation (V_{max}, V_{min}, V_{avg}, σ_V). The default is 1,000,000 for a simulation with 2 fs timestep.
ntebprep	The number of preparation biasing molecular dynamics simulation steps. This is used for system equilibration after adding the boost potential and the potential statistics (V_{max}, V_{min}, V_{avg}, σ_V) are not updated during these steps. The default is 200,000 for a simulation with 2 fs timestep.
nteb	The number of biasing molecular dynamics simulation steps. Potential statistics (Vmax, Vmin, Vavg, σV) are updated between the ntebprep and nteb steps and used to calculate the GaMD acceleration parameters, particularly E and k0. The default is 1,000,000 for a simulation with 2 fs timestep. A greater value may be needed to ensure that the potential statistics and GaMD acceleration parameters level off before running production simulation between the *nteb* and *nstlim* (total simulation length) steps. Moreover, *nteb* can be set to *nstlim*, by which the potential statistics and GaMD acceleration parameters are updated adaptively throughout the simulation. This in some cases provides more appropriate acceleration.
ntave	The number of simulation steps used to calculate the average and standard deviation of potential energies. This variable has already been used in Amber. The default is set to 50,000 for GaMD simulations. It is recommended to be updated as about 4 times of the total number of atoms in the system. Note that ntcmd and nteb need to be multiples of ntave.
sigma0P	The upper limit of the standard deviation of the total potential boost that allows for accurate reweighting if igamd is set to 1 or 3. The default is 6.0 (unit: kcal/mol).
sigma0D	The upper limit of the standard deviation of the dihedral potential boost that allows for accurate reweighting if igamd is set to 2 or 3. The default is 6.0 (unit: kcal/mol).
irest_gamd	Flag to restart GaMD simulation **= 0** (default) new simulation. A file "gamd-restart.dat" that stores the maximum, minimum, average and standard deviation of the dihedral and/or total potential energies (depending on the *igamd* flag) will be saved automatically after GaMD equilibration stage. **= 1** restart simulation (*ntcmd* is set to 0 in this case). The "gamd-restart.dat" file will be read for restart.

The GaMD algorithm is summarized as follows:

```
GaMD {
  If (irest_gamd == 0) then
    For i = 1, ..., ntcmd // run initial conventional molecular dynamics
          If (i >= ntcmdprep) Update Vmax, Vmin
          If (i >= ntcmdprep && i%ntave ==0) Update Vavg, sigmaV
    End
    Save Vmax,Vmin,Vavg,sigmaV to "gamd_restart.dat" file
    Calc_E_k0(iE,sigma0,Vmax,Vmin,Vavg,sigmaV)

    For i = ntcmd+1, ..., ntcmd+nteb // Run biasing molecular dynamics
                                    // simulation steps
          deltaV = 0.5*k0*(E-V)**2/(Vmax-Vmin)
          V = V + deltaV
          If (i >= ntcmd+ntebprep) Update Vmax, Vmin
          If (i >= ntcmd+ntebprep && i%ntave ==0) Update Vavg, sigmaV
      Calc_E_k0(iE,sigma0,Vmax,Vmin,Vavg,sigmaV)
    End
    Save Vmax,Vmin,Vavg,sigmaV to "gamd_restart.dat" file
  else if (irest_gamd == 1) then
    Read Vmax,Vmin,Vavg,sigmaV from "gamd_restart.dat" file
  End if

  For i = ntcmd+nteb+1, ..., nstlim // run production simulation
    deltaV = 0.5*k0*(E-V)**2/(Vmax-Vmin)
    V = V + deltaV
  End
}

Subroutine Calc_E_k0(iE,sigma0,Vmax,Vmin,Vavg,sigmaV) {
  if iE = 1 :
        E = Vmax
        k0' = (sigma0/sigmaV) * (Vmax-Vmin)/(Vmax-Vavg)
        k0 = min(1.0, k0')
  else if iE = 2 :
        k0'' = (1-sigma0/sigmaV) * (Vmax-Vmin)/(Vavg-Vmin)
        if 0 < k0'' <= 1 :
      k0 = k0''
      E = Vmin + (Vmax-Vmin)/k0
        else
      E = Vmax
      k0' = (sigma0/sigmaV) * (Vmax-Vmin)/(Vmax-Vavg)
      k0 = min(1.0, k0')
        end
  end
}
```

Here is a sample input file:

```
&cntrl
  imin = 0, irest = 0, ntx = 1,
  nstlim = 17000000, dt = 0.002,
  ntc = 2, ntf = 2, tol = 0.000001,
  iwrap = 1, ntb = 1, cut = 8.0,
  ntt = 3, temp0 = 300.0, tempi = 300.0,
  ntpr = 50, ntwx = 50, ntwr = 500,
  ntxo = 1, ioutfm = 1, ig = -1, ntwprt = 22,
  igamd = 3, iE = 1, irest_gamd = 0,
  ntcmd = 1000000, nteb = 1000000, ntave = 50000,
  ntcmdprep = 200000, ntebprep = 200000,
  sigma0P = 6.0, sigma0D = 6.0,
&end
```

Reweighting analysis of GaMD simulations is similar to that of previous aMD simulations, except that the *amd.log* file name needs to be replaced by *gamd.log*. Test cases have been included into the distribution of Amber, also a tutorial based on a study we performed on alanine dipeptide [395], demonstrating the usage of GaMD on unconstrained enhanced sampling and free energy calculation of biomolecules is now present on the Amber website. We encourage the user to read the paper, as well as follow the tutorial for more information. The latest updates and simulation tips of GaMD can be also found at http://gamd.ucsd.edu.

20.4. Targeted MD

The targeted MD option adds an additional term to the energy function based on the mass-weighted root mean square deviation of a set of atoms in the current structure compared to a reference structure. The reference structure is specified using the *-ref* flag in the same manner as is used for Cartesian coordinate restraints (NTR=1). Targeted MD can be used with or without positional restraints. If positional restraints are not applied (ntr=0), *sander* performs a best-fit of the reference structure to the simulation structure based on selection in *tgtfitmask* and calculates the RMSD for the atoms selected by *tgtrmsmask*. The two masks can be identical or different. This way, fitting to one part of the structure but calculating the RMSD (and thus restraint force) for another part of the structure is possible. If targeted MD is used in conjunction with positional restraints (ntr=1), only *tgtrmsmask* should be given in the control input because the molecule is ’fitted’ implicitly by applying positional restraints to atoms specified in *restraintmask*.
The energy term has the form:

```
E = 0.5 * TGTMDFRC * NATTGTRMS * (RMSD-TGTRMSD)**2
```

The energy will be added to the RESTRAINT term. Note that the energy is weighted by the number of atoms that were specified in the *tgtrmsmask* (NATTGTRMS). The RMSD is the root mean square deviation and is mass weighted. The force constant is defined using the *tgtmdfrc* variable (see below). This option can be used with molecular dynamics or minimization. When targeted MD is used, *sander* will print the current values for the actual and target RMSD to the energy summary in the output file.

itgtmd	**= 0** no targeted MD (default) **= 1** use targeted MD **= 2** use targeted MD to multiple targets (Multiply-targeted MD, or MTMD, see next section below)
tgtrmsd	Value of the target RMSD. The default value is 0. This value can be changed during the simulation by using the weight change option.
tgtmdfrc	This is the force constant for targeted MD. The default value is 0, which will result in no penalty for structure deviations regardless of the RMSD value. Note that this value can be negative, which would force the coordinates AWAY from the reference structure.
tgtfitmask	Define the atoms that will be used for the rms superposition between the current structure and the reference structure. Syntax is in Chapter 19.1.1.
tgtrmsmask	Define the atoms that will be used for the rms difference calculation (and hence the restraint force), as outlined above. Syntax is in Chapter 19.1.1.

One can imagine many uses for this option, but a few things should be kept in mind. In this implementation of targeted MD, there is currently only one reference coordinate set, so there is no way to force the coordinates to any specific structure other than the one reference. To move a structure toward a reference coordinate set, one might use an initial *tgtrmsd* value corresponding to the actual RMSD between the input and reference (*inpcrd* and *refc*). Then the weight change option could be used to decrease this value to 0 during the simulation. To move a structure away from the reference, one can increase *tgtrmsd* to values larger than zero. The minimum for this energy term will then be at structures with an RMSD value that matches *tgtrmsd*. Keep in mind that many different structures may have similar RMSD values to the reference, and therefore one cannot be sure that increasing *tgtrmsd* to a given value will result in a particular structure that has that RMSD value. In this case it is probably wiser to use the final structure, rather than the initial structure, as the reference coordinate set, and decrease *tgtrmsd* during the simulation. To address this, multiply-targeted MD is now available in Amber (*sander only*), and is described in the next section. As an additional note, a negative force constant *tgtmdfrc* can be used, but this can cause problems since the energy will continue to decrease as the RMSD to the reference increases.

Also keep in mind that phase space for molecular systems can be quite complex, and this method does not guarantee that a low energy path between initial and target structures will be followed. It is possible for the simulation to become unstable if the restraint energies become too large if a low-energy path between a simulated structure and the reference is not accessible.

Note also that the input and reference coordinates are expected to match the *prmtop* file and have atoms in the same sequence. No provision is made for symmetry; rotation of a methyl group by 120° would result in a nonzero RMSD value.

20.5. Multiply-Targeted MD (MTMD)

In Amber (*sander only*), the user may perform targeted MD calculations using multiple reference structures. Each reference may have its own associated target RMSD value and force constant, each of which can evolve independently in time. Additionally, the masks for each defined target may differ, and targeting to any given reference structure can be activated for some or part of the simulation. The energy term for MTMD is simply the sum of the energies that would be calculated for the molecule calculated relative to each target given the target RMSD and force constant for that target. The energy will then be added to the RESTRAINT term.

To use MTMD, the MTMD input file is specified using the *-mtmd* flag in the command line arguments for *sander*. The MTMD input file will contain one instance of the tgt namelist ("&tgt") for each reference structure used. The user may specify any number of reference structures.

20.5.1. Variables in the &tgt namelist:

refin	The file name of the reference structure used. The input and reference coordinates are expected to match the *prmtop* file and have atoms in the same sequence. *Default for refin is ", no reference structure given.*
mtmdform	If MTMDFORM > 0, then the reference coordinate file is formatted. Otherwise, the reference coordinate file is an unformatted (binary) file. *Default for MTMDFORM is the value assigned to MTMDFORM in the most recent namelist where MTMDFORM was specified. If MTMDFORM has not been specified in any namelist, it defaults to 1.*
mtmdstep1, mtmdstep2	Targeted MD for this structure is run for steps/iterations MTMDSTEP1 through MTMDSTEP2. If MTMDSTEP2 = 0, then TMD will be run through the end of the run, and the values of the target RMSD and the force constant will not change with time. Note that the first step/iteration is considered step 0. *Defaults for MTMDSTEP1 and MTMDSTEP2 are the values assigned to them in the most recent namelist where MTMDSTEP1 and MTMDSTEP2 were specified. If MTMDSTEP1 and MTMDSTEP2 have not been specified in any namelist, they default to 0.*
mtmdvari	If MTMDVARI $>$ 0, then the force constant and target RMSD will vary with step number. Otherwise, they are constant throughout the run. If MTMDVARI >0, then the values MTMDSTEP2, MTMDRMSD2, and MTMDFORCE2 must be specified (see below). *Default for* MTMDVARI *is the value assigned to* MTMDVARI *in the most recent namelist where* MTMDVARI *was specified. If* MTMDVARI *has not been specified in any namelist, it defaults to 0.*

mtmdrmsd, mtmdrmsd2 The target RMSD for this reference. If MTMDVARI >0, then the value of MTMDRMSD will vary between MTMDSTEP1 and MTMDSTEP2, so that, e.g. MTMDRMSD(MTMDSTEP1) = MTMDRMSD and MTMDRMSD(MTMDSTEP2) = MTMDRMSD2. *Defaults for MTMDRMSD and MTMDRMSD2 are the values assigned to them in the most recent namelist where MTMDRMSD and MTMDRMSD2 were specified. If MTMDRMSD and MTMDRMSD2 have not been specified in any namelist, they default to 0.0.*

mtmdforce, mtmdforce2 The force constant for this reference. If MTMDVARI >0, then the value of MTMDFORCE will vary between MTMDSTEP1 and MTMDSTEP2, so that, e.g. MTMDFORCE(MTMDSTEP1) = MTMDFORCE and MTMDFORCE(MTMDSTEP2) = MTMDFORCE2. *Defaults for* MTMDFORCE *and* MTMDFORCE2 *are the values assigned to them in the most recent namelist where* MTMDFORCE *and* MTMDFORCE2 *were specified. If* MTMDFORCE *and* MTMDFORCE2 *have not been specified in any namelist, they default to 0.0.*

mtmdninc If MTMDVARI >0 and MTMDNINC > 0, then the changes in the values of of MTMDRMSD and MTMDFORCE are applied as a step function, with NINC steps/iterations between each change in the target values. If MTMDNINC = 0, the change is effected continuously (at every step). *Default for MTMDNINC is the value assigned to MTMDNINC in the most recent namelist where MTMDNINC was specified. If MTMDNINC has not been specified in any namelist, it defaults to 0.*

mtmdmult If MTMDMULT=0, and the values of MTMDFORCE changes with step number, then the changes in the force constant will be linearly interpolated from MTMDFORCE→MTMDFORCE2 as the step number changes. If MTMDMULT=1 and the force constant is changing with step number, then the changes in the force constant will be effected by a series of multiplicative scalings, using a single factor, R, for all scalings. *i.e.*

```
MTMDFORCE2 = R**INCREMENTS * MTMDFORCE
```

INCREMENTS is the number of times the target value changes, which is determined by MTMDSTEP1, MTMDSTEP2, and MTMDNINC. *Default for MTMDMULT is the value assigned to MTMDMULT in the most recent namelist where MTMDMULT was specified. If MTMDMULT has not been specified in any namelist, it defaults to 0.*

mtmdmask Define the atoms that will be used for both the rms superposition between the current structure and the reference structure and the rms difference calculation (and hence the restraint force), as outlined above. Syntax is in Chapter 19.1.1.*Default for MTMDMASK is the value assigned to MTMDMASK in the most recent namelist where MTMDMASK was specified. If MTMDMASK has not been specified in any namelist, it defaults to '*', use all atoms in the fit and force calculations. *

Namelist &tgt is read for each reference structure. Input ends when a namelist statement with refin = ” (or refin not specified) is found. Note that comments can precede or follow any namelist statement, allowing comments and reference definitions to be freely mixed.

20.6. Nudged elastic band calculations

20.6.1. Background

In the nudged elastic band method (NEB),[397, 398] the path for a conformational change is approximated with a series of images of the molecule describing the path. Minimization, with the images at the endpoints fixed in space, of the total system energy provides a minimum energy path. Each image in-between is connected to the previous and next image by "springs" along the path that serve to keep each image from sliding down the energy landscape onto adjacent images. NEB is derived from the plain elastic band method, pioneered by Elber and Karplus,[399] which added the spring forces to the potential of energy surface and minimized the energy of the system. The plain elastic band method found low energy paths, but tended to cut corners in the energy landscape. NEB prevents corner cutting by truncating the spring forces in directions perpendicular to the tangent of the path.

Furthermore, the forces from the molecular potential are truncated along the path, so that images remain evenly spaced along the path. This leads to:

$$\begin{aligned} \mathbf{F} &= \mathbf{F}^{\perp} + \mathbf{F}^{\parallel} \\ \mathbf{F}^{\perp} &= -\nabla V(\mathbf{P}) + ((\nabla V(\mathbf{P}) \cdot \tau)\tau \\ \mathbf{F}^{\parallel} &= [(k_{i+1}|\mathbf{P}_{i+1} - \mathbf{P}_i| - k_i|\mathbf{P}_i - \mathbf{P}_{i-1}|) \cdot \tau]\tau \end{aligned} \tag{20.3}$$

where, if N is the number of atoms per image, $\mathbf{F}$ is the force on image i, $\mathbf{P}_i$ is the 3N dimensional position vector of image i, k_i is the spring constant between image $i-1$ and image i, V is the potential described by the force field, and τ is the 3N dimensional tangent unit vector that describes the path.

The simplest definition of τ is:

$$\tau = (\mathbf{P}_i - \mathbf{P}_{i-1})/|\mathbf{P}_i - \mathbf{P}_{i-1}| \tag{20.4}$$

This definition leads to instability in the path caused by kinks that occur where the magnitude of $\mathbf{F}^{\parallel}$ is much larger than the magnitude of $\mathbf{F}^{\perp}$. A more stable tangent definition was derived to prevent kinks in the path that depends upon the energies, E, of adjacent images.[400] The spring constants can be the same between all images or they can be scaled to move the images closer together in the regions of transition states:[401]

$$\begin{aligned} If\,(E_i > E_{ref}) \quad & then \quad k_i = k_{max} - \Delta k(E_{max} - E_i)/(E_{max} - E_{ref}) \\ & otherwise \quad k_i = k_{max} - \Delta k \end{aligned} \tag{20.5}$$

Here E_{max} is the highest energy for an image along the path, E_{ref} is the energy of the higher energy endpoint, and k_{max} and Δk are parameters with units of force per length. Because the spring force applies only in directions along the path and because the potential of the energy surface is zeroed along the path, the calculation is relatively insensitive to the magnitude of the spring constants. Care must be taken, however, to select a spring constant that does not result in higher frequency motions than those found in the system of interest.[402] At each step, before calculating the spring forces that compose $\mathbf{F}^{\parallel}$, each image's neighbor is rotated and translated onto itself to find the RMSD minimum, based on a subset of the system's atoms which the user can define. In this way, each image remains a continuous MD simulation, and the communication of coordinates can be greatly reduced.

Energy minimization of the path is complicated by the fact that the forces are truncated according to the tangent direction, making it impossible to define a Lagrangian.[402] Conjugate gradient minimization, therefore, cannot be used to find the minimum energy path. An algorithm for quenched molecular dynamics has been used to find the minimum.[398] With this method, the component of the velocity parallel to the force is kept, but perpendicular components are scaled:

$$\begin{aligned} If\,(\mathbf{v} \cdot \mathbf{f} > 0) \quad & then \quad \mathbf{v} = (\mathbf{v} \cdot \mathbf{f})\mathbf{f} \\ & otherwise \quad \mathbf{v} = x(\mathbf{v} \cdot \mathbf{f})\mathbf{f} \end{aligned} \tag{20.6}$$

where $\mathbf{f}$ is the 3N-dimensional unit force vector, $\mathbf{v}$ is the 3N-dimensional velocity vector, and x is a scaling factor less than one. Recently, a super-linear minimization method was described using an adopted basis Newton-Raphson minimizer.[402]

A partial NEB (PNEB) implementation is available, and is the only form of NEB that is currently supported [403]. This implementation allows the NEB method to be applied to a user defined subset of the system of interest. It requires users to define the part of the system to apply NEB force decoupling to, as well as the part of the system to RMS fit neighboring images to, to remove rotational and translational motion. This allows NEB to be used efficiently in large systems where a local transition is desired, or in explicitly solvated systems.

As with the previous implementation of NEB in *sander.MPI* [404], minimization of the system energies along the lowest potential energy path is achieved by simulated annealing. This requires no hypothesis for a starting path, but does require careful judgment of the temperature and length of time required to populate the minimum energy path. The initial coordinates can have multiple copies of the structure superimposed on the start and endpoints. When adjacent structures are superimposed, the tangent, τ is 0 in every direction. This case is explicitly handled so that the calculation is stable.

20.6.2. Preparing input files for NEB

The NEB capability is implemented inside *sander.MPI* and uses the multisander functionality. Input *prmtop* and *inpcrd* files for NEB should be generated using *Leap*. For NEB you need as a minimum a *prmtop* for a single image of your molecule and two *inpcrd* files representing each end of the pathway.

The following are some notes for preparing NEB input files:

1. Always check that the *prmtop* files generated for the endpoint coordinates are identical. This can be done by diffing the files. One *prmtop* must be used to describe both endpoints' *inpcrd* files.
2. If you have intermediate structures along the path, you must make sure the *prmtop* is appropriate for this structure as well.
3. The endpoint images serve as coordinate reference points, and the first and last images remain fixed in coordinate and energy space along the path. No simulation is performed on these during NEB optimization, so they must initially be well minimized to prevent the rest of the images from migrating to a local minimum before the conformational transition occurs. Take this into consideration when choosing the number of images to connect along the path.

Multisander requires a groupfile input, where each line of the groupfile is the sander command for each individual image's MD simulation. Multiple copies of each endpoint image are used for the initial simulation. If intermediates are available and the user wishes to include them, they should be added sequentially in between the endpoint conformations in the order in which these structures are thought to appear along the transition path.

Notes for running NEB using multisander:

1. The number of CPUs specified must be a multiple of the number of images. You can run this on a standard desktop computer, but it will generally be more efficient to run it on a minimum of one processor per image.
2. If the user has access to parallel computing resources, multiple processors per image may be used. Careful benchmarking should be done to gauge the best balance between computational efficiency in calculating the dynamics of each image and slowdown caused by communications overhead at each step.

20.6.3. Input Variables

ineb	Flag for nudged elastic band. A value of 0 (default) means that no nudged elastic band will be used. A value of 1 means that a NEB simulation is being performed.
tgtfitmask	Flag which sets atoms to RMS fit each image's neighbor to itself. This must not include solvent, which, due to diffusion, overlapping proves impossible. The more atoms you choose, the more communication has to be done by each bead. Syntax for this is here: 19.1.1
tgtrmsmask	Flag which sets atoms to decouple NEB forces for PNEB. This can be set to all atoms of the solute, or a subset of atoms which best describes the area of the system which undergoes the conformational change you wish to see. Syntax for this is here: 19.1.1
skmax	Spring constant or kmax from above (100 by default).
skmin	If skmin = skmax, a fixed spring constant is used. Otherwise, skmin is taken from above for scaled spring constants (50 by default).
tmode	If 1 (default), use the revised tangent definition that prevents kinks. For any other value, use the simple (original) tangent definition.
vv	If this is 1, use the quenched velocity Verlet minimization; otherwise, do not.
vfac	Scaling factor for quenched velocity Verlet algorithm. (0.0 by default).

Sample input file for running initial heating along the path.

Below is an example input file that can be used to perform the initial heating step of a NEB run. Note that the input and topology files must be identical for each bead; the names of the output, trajectory, restart and info files should not be the same between beads.

```
Alanine NEB initial MD with small K
&cntrl
  imin = 0, irest = 0,
  ntc=1, ntf=1,
  ntpr=1, ntwx=500,
  ntb = 0, cut = 999.0, rgbmax=999.0,
  igb = 1, saltcon=0.2,
  nstlim = 40000, nscm=0,
  dt = 0.0005, ig=42,
  ntt = 3, gamma_ln=1000.0,
  tempi=0.0, temp0=300.0,
  tgtfitmask=":1,2,3",
  tgtrmsmask=":1,2,3@N,CA,C",
  ineb = 1,skmin = 10,skmax = 10,
  nmropt=1,
/
&wt type='TEMP0', istep1=0,istep2=35000,
  value1=0.0, value2=300.0
/
&wt type='END'
/
```

The important NEB specific lines are shown in boldface type. **tgtfitmask** variable denotes the atoms that will be used to RMS fit each bead onto its neighbor images at each step. In this case all atoms of residues 1 2 and 3 are specified. The **tgtrmsmask** variable denotes the atoms that the NEB forces will be applied to. In this case the backbone atoms of residues 1, 2 and 3 are specified. In general, the atoms that have NEB forces applied to them should be those involved in the transition of interest. If the specific transition is not known or there are many degrees of freedom involved in the transition, one can simply specify all solute atoms. It is *not* recommended to apply NEB forces to solvent atoms. For more examples, please refer to the runs in the *$AMBERHOME/test/neb-testcases* directory, or see reference [403].

20.6.4. Important Considerations for NEB Simulations

With the implementation of PNEB, it is important to understand some limitations of the method. Only part of the system is simulated with NEB forces, indicating this part of the system is moving along the minimum potential energy landscape of the transition path. However, the part of the system to which NEB is not applied is not necessarily forced along this minimum potential energy path, and attention must be paid to the convergence of this part of the system. The conformational change in this part of the system is no doubt accelerated, since it responds to the part of the system to which NEB forces are applied. Further equilibration of the system may need to be done if the user wishes to examine changes not local to the area the NEB forces are applied to.

Careful attention must be paid to optimization methods, to assure that conformational space is explored for the NEB part of the system, while the integrity of the non-NEB part remains intact. As in all NEB implementations, a general caveat is that as the system size increases, the degrees of freedom increase and conformational changes become more difficult to quantify. While NEB is a method which does not necessitate a reaction coordinate, care should be taken when analyzing the resulting minimum energy path. It is reccommended that NEB be run a statistically relevant number of times so reproducibility (and convergence) of the minimum energy path can be studied.

20.7. Low-MODe (LMOD) methods

István Kolossváry's LMOD methods for minimization, conformational searching, and flexible docking[405–408] are now fully implemented in Amber. The centerpiece of LMOD is a conformational search algorithm based on eigenvector following of low frequency vibrational modes. It has been applied to a spectrum of computational chemistry domains including protein loop optimization and flexible active site docking.

Details of the LMOD procedure, and hints on getting good performance, are given Section 39.5, which should be consulted before trying the procedures in *sander*. The only difference between the implementation in *sander* and in *NAB* involves details of how the input is specified; the same LMOD code is linked into both. The sections below give input details for *sander*.

New for AmberTools17: there are now **four "real-life" examples** of carrying out LMOD searches: look in *$AMBERHOME/AmberTools/examples/nab/lmod_**. Each directory has a README file that give more information.

20.7.1. XMIN

The XMIN methods for minimization are traditional and manifold in the field of unconstrained optimization: PRCG is a Polak-Ribiere nonlinear Conjugate Gradient algorithm,[409] LBFGS is a Limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton algorithm,[410] and TNCG is a Truncated Newton linear Conjugate Gradient method with optional LBFGS preconditioning.[411]

Some of the &cntrl namelist variables that control Amber's other minimization facilities also control XMIN. Consequently, non-experts can employ the default XMIN method merely by specifying ntmin = 3.

maxcyc The maximum number of cycles of minimization. Default is 1 to be consistent with Amber's other minimization facilities although it may be unrealistically short.

ntmin The flag for the method of minimization.

= 3 The XMIN method is used.

= 4 The LMOD method is used. The LMOD procedure employs XMIN for energy relaxation and minimization.

drms The convergence criterion for the energy gradient: minimization will halt when the root-mean-square of the Cartesian elements of the gradient is less than DRMS. Default is 1.0E-4 kcal/mole Åto be consistent with Amber's other minimization facilities although it may be unrealistically strict.

Other options that control XMIN are in the scope of the &lmod namelist. These parameters enable expert control of XMIN.

lbfgs_memory_depth The depth of the LBFGS memory for LBFGS minimization, or LBFGS preconditioning in TNCG minimization. Default is 3. Suggested alternate value is 5. The value 0 turns off LBFGS preconditioning in TNCG minimization.

matrix_vector_product_method The finite difference Hv matrix-vector product method: "forward" = forward difference, "central" = central difference. Default is forward difference.

xmin_method The minimization method: "PRCG" = Polak-Ribiere Conjugate Gradient, "LBFGS" = Limited-memory Broyden-Fletcher-Goldfarb-Shanno, and "TNCG" = Optionally LBFGS-preconditioned Truncated Newton Conjugate Gradient. Default is LBFGS.

xmin_verbosity The verbosity of the internal status output from the XMIN package: 0 = none, 1 = minimization details, and 2 = minimization and line search details plus CG details in TNCG. Currently, the XMIN status output may be disordered with respect to Amber's output. Default is 0, no output of the XMIN package internal status. Note that XMIN is also available in AmberTools, in the NAB package. An annotated example output corresponding to XMIN_VERBOSITY=2 can be found in the NAB documentation.

20.7.2. LMOD

Some of the options that control LMOD have the same names as Amber's other minimization facilities. See the XMIN section immediately above. Other options that control LMOD are in the scope of the &lmod namelist. These parameters enable expert control of LMOD.

arnoldi_dimension The dimension of the ARPACK Arnoldi factorization. Zero specifies the whole space, that is, three times the number of atoms. Default is 0, the whole space. Basically, the ARPACK package used for the eigenvector calculations solves multiple "small" eigenvalue problems instead of a single "large" problem, which is the diagonalization of the three times the number of atoms by three times the number of atoms Hessian matrix. This parameter is the user specified dimension of the "small" problem. The allowed range is total_low_modes + 1 <= arnoldi_dimension <= three times the number of atoms. The default means that the "small" problem and the "large" problem are identical. This is the preferred, i.e., fastest, calculation for small to medium size systems, because ARPACK is guaranteed to converge in a single iteration. The ARPACK calculation scales with three times the number of atoms times the arnoldi_dimension squared and, therefore, for larger molecules there is an optimal arnoldi_dimension much less than three times the number of atoms that converges much faster in multiple iterations (possibly thousands or tens of thousands of iterations). The key to good performance is to select an arnoldi_dimension such that all the ARPACK storage fits in memory. For proteins, arnoldi_dimension=1000 is generally a good value, but often a very small 50-100 Arnoldi dimension provides the fastest net computational cost with very many iterations.

conflib_filename The user-given filename of the LMOD conformational library. The file format is standard Amber trajectory file. The conformations are stored in energetic order (global minimum energy structure first), the number of conformations <= conflib_size. The default filename is *conflib*.

conflib_size The number of conformations to store in conflib. Default is 3.

energy_window The energy window for conformation storage; the energy of a stored structure will be in the interval [global_min, global_min + energy_window]. Default is 0, only storage of the global minimum structure.

explored_low_modes The number of low frequency vibrational modes used per LMOD iteration. Default is 3.

frequency_eigenvector_recalc The frequency, measured in LMOD iterations, of the recalculation of eigenvectors. Default is 3.

frequency_ligand_rotrans The frequency, measured in LMOD iterations, of the application of rigid-body rotational and translational motions to the ligand(s). At each frequency_ligand_rotrans-th LMOD iteration number_ligand_rotrans rotations and translations are applied to the ligand(s). Default is 1, ligand(s) are rotated and translated at every LMOD iteration.

lmod_job_title The user-given title for the job that goes in the first line of the conflib and lmod_trajectory files. The default job title is "job_title_goes_here".

lmod_minimize_grms The gradient RMS convergence criterion of structure minimization. Default is 0.1.

lmod_relax_grms The gradient RMS convergence criterion of structure relaxation. Default is 1.0.

lmod_restart_frequency The frequency, in LMOD iterations, of conflib updating and LMOD restarting with a randomly chosen structure from the pool. Default is 5.

lmod_step_size_max The maximum length of a single LMOD ZIG move. Default is 5.0 Å.

lmod_step_size_min The minimum length of a single LMOD ZIG move. Default is 2.0 Å.

lmod_trajectory_filename The filename of the LMOD pseudo trajectory. The file format is standard Amber trajectory file. The conformations in this file show the progress of the LMOD search. The number of conformations = number_lmod_iterations + 1. The default filename is *lmod_trajectory*.

lmod_verbosity The verbosity of the internal status output from the LMOD package: 0 = none, 1 = some details, 2 = more details, 3 = everything including ARPACK information. Currently, the LMOD status output may be disordered with respect to Amber's output. Default is 0, no output of the LMOD package internal status. Note that LMOD is also available in AmberTools, in the NAB package. An annotated example output corresponding to LMOD_VERBOSITY=2 can be found in the NAB documentation.

monte_carlo_method The Monte Carlo method: "Metropolis" = Metropolis Monte Carlo, "Total_Quench" = the LMOD trajectory always proceeds towards the lowest lying neighbor of a particular energy well found after exhaustive search along all of the low modes, and "Quick_Quench" = the LMOD trajectory proceeds towards the first neighbor found, which is lower in energy than the current point on the path, without exploring the remaining modes. Default is Metropolis Monte Carlo.

number_free_rotrans_modes The number of rotational and translational degrees of freedom. This is related to the number of frozen or tethered atoms in the system: 0 atoms dof=6, 1 atom dof=3, 2 atoms dof=1, >=3 atoms dof=0. Default is 6, no frozen atoms.

number_ligand_rotrans The number of rigid-body rotational and translational motions applied to the ligand(s). Such applications occur at each frequency_ligand_rotrans-th LMOD iteration. Default is 0, no rigid-body motions applied to the ligand(s).

number_ligands The number of ligands for flexible docking. Default is 0, no ligand(s).

number_lmod_iterations The number of LMOD iterations. Default is 10. Note that setting number_lmod_iterations = 0 will result in a single energy minimization.

number_lmod_moves The number of LMOD ZIG-ZAG moves. Zero means that the number of ZIG-ZAG moves is not pre-defined, instead LMOD will attempt to cross the barrier in as many ZIG-ZAG moves as it is necessary. The criterion of crossing an energy barrier is stated above in the "LMOD Procedure" background section. number_lmod_moves $>$ 0 means that multiple barriers may be crossed and LMOD can carry the molecule to a large distance on the potential energy surface without severely distorting the geometry. Default is 0, LMOD will determine automatically where to stop the ZIG-ZAG sequence.

random_seed The seed of the random number generator. Default is 314159.

restart_pool_size The size of the pool of lowest-energy structures to be used for restarting. Default is 3.

rtemperature The value of RT in Amber energy units. This is utilized in the Metropolis criterion. Default is 1.5.

total_low_modes The total number of low frequency vibrational modes to be used. Default is the minimum of 10 and three times the number of atoms minus the number of rotational and translational degrees of freedom (number_free_rotrans_modes).

The following commands are part of the &lmod namelist. These commands control the way LMOD applies explicit translations and rotations to one or more ligands and take effect only if number_ligands >= 1. All commands are lists in square brackets, separated by commas such as [1, 33, 198], however, the list is read by Sander as a string and, therefore, it should be enclosed in single quotes.

ligstart_list, ligend_list The serial number(s) of the first/last atom(s) of the ligand(s). Type integer. The number(s) should correspond to the numbering in the Amber input files prmtop and inpcrd/restart. For example, if there is only one ligand and it starts at atom 193, the command should be ligstart_list = '[193]'. If there are three ligands, the command should be, e.g., '[193, 244, 1435]'. The same format holds for all of the following commands. Note that the ligand(s) can be anywhere in the atom list, however, a single ligand must have continuous numbering between the corresponding ligstart_list and ligend_list values. For example, ligstar_list = '[193, 244, 1435]' and ligend_list = '[217, 302, 1473]'.

ligcent_list The serial number(s) of the atom(s) of the ligand(s), which serves as the center of rotation. Type integer. The value zero means that the center of rotation will be the geometric center of gravity of the ligand.

rotmin_list, rotmax_list The range of random rotation of a particular ligand about the origin defined by the corresponding ligcent_list value is specified by the commands rotmin_list and rotmax_list. The angle is given in +/- degrees. Type float. For example, in case of a single ligand and ligcent_list = '[0]', rotmin_list = '[30.0]' and rotmax_list = '[180.0]' means that random rotations by an angle +/- 30-180 degrees about the center of gravity of the ligand, will be applied. Similarly, with number_ligands = 2, ligcent_list= 120.0]' means that the first ligand will be rotated like in the single ligand example in this paragraph, but a second ligand will be rotated about its atom number 201, by an angle +/- 60-120 degrees.

trmin_list, trmax_list The range of random translation(s) of ligand(s) is defined by the same way as rotation. For example, with number_ligand = 1, trmin_list = '[0.1]' and trmax_list = '[1.0]' means that a single ligand is translated in a random direction by a random distance between 0.1 and 1.0 Angstroms.

21. Free energies

21.1. Thermodynamic integration

In a free energy calculation, the system evolves according to a mixed potential (such as in Eqs. 21.3 or 21.4, below). The essence of free energy calculations is to record and analyze the fluctuations in the values of V_0 and V_1 (that is, what the energies *would have been* with the endpoint potentials) as the simulation progresses. For thermodynamic integration (which is a very straightforward form of analysis) the required averages can be computed "on-the-fly" (as the simulation progresses), and printed at the end of a run. For more complex analyses (such as the Bennett acceptance ratio scheme), one needs to write the history of the values of V_0 and V_1 to a file, and later post-process this file to obtain the final free energy estimates.

There is not room here to discuss the theory of free energy simulations, and there are many excellent discussions elsewhere.[9, 412, 413] There are also plenty of recent examples to consult.[414, 415] Such calculations are demanding, both in terms of computer time, and in a level of sophistication to avoid pitfalls that can lead to poor convergence. Since there is no one "best way" to estimate free energies, *sander* and *pmemd* primarily provide the tools to collect the statistics that are needed. Assembling these into a final answer, and assessing the accuracy and significance of the results, generally requires some calculations outside of what Amber provides, *per se*. The discussion here will assume a certain level of familiarity with the basis of free energy calculations.

Both *sander* and *pmemd* have the capability of doing simple thermodynamic free energy calculations, using either PME or generalized Born potentials. When *icfe* is set to 1, information useful for doing thermodynamic integration estimates of free energy changes will be computed. The implementation is different between *sander* and *pmemd*. For *sander*, you must use the *multisander* capability to create two groups, one corresponding to the starting state, and a second corresponding to the ending state (see Section 17.11 for information); you will need a *prmtop* file for each of these two endpoints. For *pmemd*, you use a single *prmtop* file which contains both the starting and ending states. For both *sander* and *pmemd* a mixing parameter λ is used to interpolate between the "unperturbed" and "perturbed" potential functions.

21.1.1. Thermodynamic integration using Sander

There are now two different ways to prepare a thermodynamic integration free energy calculation in Sander. The first is unchanged from previous versions of Amber: Here, the two prmtop files that you create must have the same number of atoms, and the atoms must appear *in the same order* in the two files. This is because there is only one set of coordinates that are propagated in the molecular dynamics algorithm. If there are more atoms in the initial state than in the final, "dummy" atoms must be introduced into the final state to make up the difference. Although there is quite a bit of flexibility in choosing the initial and final states, it is important in general that the system be able to morph "smoothly" from the initial to the final state. Alternatively, you can set up your system to use the softcore potential algorithm described below. This will remove the requirement to prepare "dummy" atoms and allows the two *prmtop* files to have different numbers of atoms.

The basics of the *multisander* functionality are given in Section 17.11, but the mechanics are really quite simple. You start a free energy calculation as follows:

```
mpirun -np 4 sander.MPI -ng 2 -groupfile <filename>
```

Since there are 4 total cpu's in this example, each of the two groups will run in parallel with 2 cpu's each. The number of processors must be a multiple of two. The *groups* file might look like this:

```
-O -i mdin -p prmtop.0 -c eq1.x -o md1.o -r md1.x -inf mdinfo
-O -i mdin -p prmtop.1 -c eq1.x -o md1b.o -r md1b.x -inf mdinfob
```

The input (*mdin*) and starting coordinate files must be the same for the two groups. Furthermore, the two *prmtop* files must have the same number number of atoms, in the same order (since one common set of coordinates will be used for both.) The simulation will use the masses found in the first *prmtop* file; in classical statistical mechanics, the Boltzmann distribution in coordinates is independent of the masses so this should not represent any real restriction.

On output, the two restart files should be identical, and the two output files should differ only in trivial ways such as timings; there should be no differences in any energy-related quantities, except if energy decomposition is turned on (*idecomp* > 0); then only the output file of the first group contains the per residue contributions to $\langle \partial V / \partial \lambda \rangle$. For our example, this means that one could delete the *md1b.o* and *md1b.x* files, since the information they contain is also in *md1.o* and *md1.x*. (It is a good practice, however, to check these file identities, to make sure that nothing has gone wrong.)

21.1.2. Thermodynamic integration using PMEMD

In *pmemd*, there is only a single input topology file which contains the atoms corresponding to both the start and end states. As explained in Ref. [416] this removes redundant calculations, greatly improving the efficiency of the code. In order to accommodate these changes, some input flags have been modified compared to *sander*. These are marked in the sections below. Also, simulations at the endpoints, $\lambda = 0$ or $\lambda = 1$, will work even for soft core simulations.

The *prmtop* file needs to be carefully prepared in order to be compatible with the *pmemd* TI implementation. A number of examples for setting up the *prmtop* file are given below in section 21.1.6. This is not a complete tutorial on TI calculations, but explains how to prepare the new *prmtop* format for various types of TI calculations.

Performance of the PME TI pmemd implementation is approximately 75% that of a regular PME MD simulation with roughly the same parallel scaling. The difference in absolute performance comes from the fact that a PME calculation is not pairwise decomposable and therefore the reciprocal space calculation needs to be carried out twice per time step, once for V_0 and V_1. For GB TI the performance difference is approximately 50% since the GB radii calculation is not pairwise decomposable and thus two non-bond calculations are carried out per time step.

The exception to this performance difference is when one is running just vdW only soft core transformations. In this situation there are no charges on the TI atoms and thus the charges for all of the atoms in both V_0 and V_1 are the same. Hence the long range electrostatics calculation only needs to be done once per step, rather than twice (for V_0 and V_1) . This results in performance roughly equivalent to a standard MD simulation. This optimization is determined automatically and can be seen in the mdout file – 'No charge on TI atoms. Skipping extra recip sum.' To determine the total free energy change it is neccessary to carry out additional simulations to determine the free energy of removing the charges from the molecules. It is up to the user to decide which path through the thermodynamic cycle will be more efficient for their system of interest.

21.1.3. Basic inputs for thermodynamic integration

icfe	The basic flag for free energy calculations. The default value of 0 skips such calculations. Setting this flag to 1 turns them on, using the mixing rules in Eq. 21.3, below.
clambda	The value of λ for this run, as in Eqs. 21.3 and 21.4, below. Zero corresponds to the unperturbed Hamiltonian V_0. λ=1 corresponds to the perturbed Hamiltonian V_1.
klambda	The exponent in Eq. 21.4, below.
tishake	Flag that determines how SHAKE is handled:

= 0 Coordinates are synchronized after SHAKE, no constraints removed (default).

= 1 SHAKE is removed between bonds containing one common and one unique atom. This was the default in previous versions of *sander.* Note that disabling SHAKE requires the use of a 1 fs timestep.

21.1.3.1. Input flags specific to Sander

idecomp	Flag that turns on/off decomposition of $\langle \partial V/\partial\lambda \rangle$ on a per-residue level. The default value of 0 turns off energy decomposition. A value of 1 turns the decomposition on, and 1-4 nonbonded energies are added to internal energies (bond, angle, torsional). A value of 2 turns the decomposition on, and 1-4 nonbonded energies are added to EEL and VDW energies, respectively. The frequency by which values of $\langle \partial V/\partial\lambda \rangle$ are included into the decomposition is determined by the NTPR flag. This ensures that the sum of all contributions equals the average of all total $\langle \partial V/\partial\lambda \rangle$ values output every NTPR steps. All residues, including solvent molecules, have to be chosen by the RRES card to be considered for decomposition. The RES card determines which residue information is finally output. The output comes at the end of the *mdout* file. For each residue contributions of internal -, VdW-, and electrostatic energies to $\langle \partial V/\partial\lambda \rangle$ are given as an average over all (NSTLIM/NTPR) steps. In a first section total per residue values are output followed below by further decomposed values from backbone and sidechain atoms.

21.1.3.2. Input flags specific to PMEMD

timask1	Specifies the atoms unique to V_0 in ambmask format.
timask2	Specifies the atoms unique to V_1 in ambmask format.

21.1.4. Background theory of thermodynamic integration

The *sander and pmemd* programs do not compute free energies; it is up to the user to combine the output of several runs (at different values of λ) and to numerically estimate the integral:

$$\Delta A = A(\lambda = 1) - A(\lambda = 0) = \int_0^1 \langle \partial V/\partial\lambda \rangle_\lambda \, d\lambda \tag{21.1}$$

If you understand how free energies work, this should not be at all difficult. However, since the actual values of λthat are needed, and the exact method of numerical integration, depend upon the problem and upon the precision desired, we have not tried to pre-code these into the program.

The simplest numerical integration is to evaluate the integrand at the midpoint:

$$\Delta A \simeq \langle \partial V/\partial\lambda \rangle_{1/2}$$

This might be a good first thing to do to get some picture of what is going on, but is only expected to be accurate for very smooth or small changes, such as changing just the charges on some atoms. Gaussian quadrature formulas of higher order are generally more useful:

$$\Delta A = \sum_i w_i \langle \partial V/\partial\lambda \rangle_i \tag{21.2}$$

Some weights and quadrature points are given in the accompanying table; other formulas are possible,[417] but the Gaussian ones listed there are probably the most useful. The formulas are always symmetrical about λ= 0.5, so that λ and $(1-\lambda)$ both have the same weight. For example, if you wanted to use 5-point quadrature, you would need to run five jobs, setting $\lambda$to 0.04691, 0.23076, 0.5, 0.76923, and 0.95308 in turn. (Each value of λ should have an equilibration period as well as a sampling period; this can be achieved by setting the *ntave* parameter.) You would then multiply the values of $\langle \partial V/\partial\lambda \rangle_i$ by the weights listed in the Table, and compute the sum.

When *icfe=1* and *klambda* has its default value of 1, the simulation uses the mixed potential function:

$$V(\lambda) = (1-\lambda)V_0 + \lambda V_1 \tag{21.3}$$

where V_0 is the potential with the original Hamiltonian, and V_1 is the potential with the perturbed Hamiltonian. The program also computes and prints $\langle \partial V/\partial\lambda \rangle$ and its averages; note that in this case, $\langle \partial V/\partial\lambda \rangle = V_1 - V_0$. This is referred to as linear mixing, and is often what you want unless you are making atoms appear or disappear. If some of the perturbed atoms are "dummy" atoms (with no van der Waals terms, so that you are making these atoms "disappear" in the perturbed state), the integrand in Eq. 21.1 diverges at λ= 1; this is a mild enough divergence that the overall integral remains finite, but it still requires special numerical integration techniques to obtain a good estimate of the integral.[413] *Sander and pmemd* implement one simple way of handling this problem: if you set *klambda* > 1, the mixing rules are

$$V(\lambda) = (1-\lambda)^k V_0 + [1-(1-\lambda)^k] V_1 \tag{21.4}$$

where k is given by *klambda*. Note that this reduces to Eq. 21.3 when k = 1, which is the default. If $k \geq 4$, the integrand remains finite as $\lambda \to 1$.[413] We have found that setting k= 6 with disappearing groups as large as tryptophan works, but using the softcore option (*ifsc*>0) instead is generally preferred.[418] Note that the behavior of $\langle \partial V/\partial\lambda \rangle$ as a function of λ is not monotonic when *klambda* > 1. You may need a fairly fine quadrature to get converged results for the integral, and you may want to sample more carefully in regions where $\langle \partial V/\partial\lambda \rangle$ is changing rapidly.

Notes:

1. This is implemented in *sander* by calling the force() routine independently for each *multisander* group and then combining the forces on each step. For a fixed number of processors this increases the cost of the calculation compared with the *pmemd* code, which only calculates the differences between V_0 and V_1.
2. It is rather easy to make mistakes when running TI calculations. It is generally good to carry out a short run (say 50 steps) setting ntpr=1. Then check the following; if either test fails, be sure to fix the problem before proceeding.
    a) The restart files from V_0 and V_1 should be identical for *sander* (for *pmemd* there will only be a single restart file).
    b) If you diff the output files for *sander*, there should only be simple differences (for *pmemd* there will only be a single combined output file). All energies, temperatures, pressures, etc. should be the same in the two files. Simulations with *sander* using the QM/MM facility may show differences in the SCF energies, but be sure that the total energies, and all the MM components, are the same.
3. Eq. 21.4 is designed for having dummy atoms in the perturbed Hamiltonian, and "real" atoms in the regular Hamiltonian. You must ensure that this is the case when you set up the system in LEaP. (See the softcore section, below, for a more general way to handle disappearing atoms, which does not require dummy atoms at all.)
4. One common application of this model is to pKa calculations, where the charges are mutated from the protonated to the deprotonated form. Since H atoms bonded to oxygen already have zero van der Waals radii (in the Amber force fields and in TIP3P water), once their charge is removed (in the deprotonated form) they are really then like dummy atoms. For this special situation, there is no need to use *klambda* > 1: since the van der Waals terms are missing from both the perturbed and unperturbed states, the proton's position can never lead to the large contributions to $\langle \partial V/\partial\lambda \rangle$ that can occur when one is changing from a zero van der Waals term to a finite one.
5. The implementation requires that the masses of all atoms be the same on all threads. To enforce this, the masses found for V_0 are used for V_1 as well. In classical statistical mechanics, the canonical distribution of configurations (and hence of potential energies) is unaffected by changes in the masses, so this should not pose a limitation. Since the masses for V_1 are ignored, they do not have to match those found for V_0.
6. Special care needs to be taken when using SHAKE for atoms whose force field parameters differ in the two end points. The same bonds must be SHAKEN in both cases, and the equilibrium bond lengths must also be the same. By default, the coordinates from V_0 are synchronized with those from V_1 after SHAKE. This will work for small perturbations, but if there is a significant change in bond length, it may be necessary to use the *noshakemask* input to remove SHAKE from the regions that are being perturbed. If this is done, be sure to set *tishake*=1 and to use a 1 fs timestep. Special care needs to be taken when water molecules are part of the region that is changing. You need to make sure that the “number of 3-point waters” is the same in both V_0 and V_1. This may require setting *jfastw* and/or building the structure so that *sander or pmemd* do

not think that the water molecules involved are actually rigid waters. Also, just setting *noshakemask* might not be enough, since this flag does not affect the *settle* routine that handles rigid waters.

n	λ_i	$1-\lambda_i$	w_i
1	0.5		1.0
2	0.21132	0.78867	0.5
3	0.1127	0.88729	0.27777
	0.5		0.44444
5	0.04691	0.95308	0.11846
	0.23076	0.76923	0.23931
	0.5		0.28444
7	0.02544	0.97455	0.06474
	0.12923	0.87076	0.13985
	0.29707	0.70292	0.19091
	0.5		0.20897
9	0.01592	0.98408	0.04064
	0.08198	0.91802	0.09032
	0.19331	0.80669	0.13031
	0.33787	0.66213	0.15617
	0.5		0.16512
12	0.00922	0.99078	0.02359
	0.04794	0.95206	0.05347
	0.11505	0.88495	0.08004
	0.20634	0.79366	0.10158
	0.31608	0.68392	0.11675
	0.43738	0.56262	0.12457

Table 21.1.: *Abscissas and weights for Gaussian integration.*

21.1.5. Softcore Potentials in Thermodynamic Integration

Softcore potentials provide an additional way to perform thermodynamic integration calculations in Amber. The system setup has been simplified so that appearing and disappearing atoms can be present at the same time and no dummy atoms need to be introduced. For *sander*, two prmtop files, corresponding to the start and end states (V_0 and V_1) of the desired transformation need to be used. The common atoms that are present in both states need to appear in the same order in both prmtop files and must have identical starting positions. In addition to the common atoms, each process can have any number of unique soft core atoms, as specified by scmask. For *pmemd*, a single prmtop file is used, containing the unique atoms for both the start and end states. The soft core atoms are specified by scmask1 and scmask2 for V_0 and V_1 respectively.

A modified version of the vdW equation is used to smoothly switch off non-bonded interactions of these atoms with their common atom neighbors:

$$V_{V_0,disappearing} = 4\varepsilon(1-\lambda)\left[\frac{1}{\left[\alpha\lambda+(\frac{r_{ij}}{\sigma})^6\right]^2} - \frac{1}{\alpha\lambda+(\frac{r_{ij}}{\sigma})^6}\right] \tag{21.5}$$

$$V_{V_1,appearing} = 4\varepsilon\lambda\left[\frac{1}{\left[\alpha(1-\lambda)+(\frac{r_{ij}}{\sigma})^6\right]^2} - \frac{1}{\alpha(1-\lambda)+(\frac{r_{ij}}{\sigma})^6}\right] \tag{21.6}$$

Please refer to Ref [418] for a description of the implementation and performance testing when compared to the TI methods described above using *sander*. For similar information pertaining to pmemd please see Ref [416]. Note that the term “disappearing” is used here, but it would probably be better to say that atoms present in V_0 but not in V_1 are “decoupled” from their environment: the interactions among the “disappearing” atoms are not changed, and do not contribute to $\langle \partial V/\partial \lambda \rangle$. If the disappearing atoms are a separate molecule (say a non-covalently-bound ligand), this can be viewed as a transfer to the gas-phase.

Note that a slightly different setup is required for using soft core potentials compared to older TI-implementations. Specifically, the difference is that to add or remove atoms without soft core potentials, they are transformed into interactionless dummy particles, so both end state prmtop files have the same number of atoms. When using soft core potentials instead, no dummy atoms are needed and the end states should be built without them. Therefore prmtop files for non soft core simulations may have to be adapted to be used with soft core potentials and vice versa.

All bonded interactions of the unique atoms are recorded separately in the output file (see below). Any bond, angle, dihedral or 1-4 term that involves at least one appearing or disappearing atom is not scaled by λ and does not contribute to $\langle \partial V/\partial \lambda \rangle$. Therefore, output from both processes will not be identical when soft core potentials are used. Softcore transformations avoid the origin singularity effect and therefore linear mixing can (and should) always be used with them. Since the unique atoms become decoupled from their surroundings at high or low lambdas and energy exchange between them and surrounding solvent becomes inefficient, a Berendsen type thermostat should not be used for SC calculations. Unlike in previous versions, SHAKE constraints are not automatically removed from bonds between common and unique atoms. Instead, the coordinates corresponding to common atoms in $V_0$are synchronized with those of V_1. The original behavior can be restored using *tishake*. The icfe and klambda parameters should be set to 1 for a soft core run and the desired lambda value will be specified by clambda. When using softcore potentials with *sander*, λ values should be picked so that $0.01 < \text{clambda} < 0.99$. The *pmemd* implementation allows lambda to be set to any value between 0.0 and 1.0, thus simulations at the endpoints are possible.

Additionally, the following parameters are available to control the TI calculation:

ifsc	Flag for soft core potentials **= 0** SC potentials are not used (default) **= 1** SC potentials are used. Be sure to use prmtop files that are suitable for this, i.e. not-containing dummy atoms (see above)
scalpha	The α parameter in 21.5 and 21.6, its default value is 0.5. Other values have not been extensively tested
logdvdl	If set to .ne. 0, a summary of all $\partial V/\partial \lambda$ values calculated during every step of the run will be printed out at the end of the simulation for postprocessing.
dvdl_norest	This option is now deprecated. Restraints involving soft core atoms are now decoupled from the rest of the system. The energy is listed separately and does not contribute to $\partial V/\partial \lambda$.
dynlmb	If set to a value .gt. zero, clambda is increased by dynlmb every ntave steps. This can be used to perform simulations with dynamically changing lambdas.
crgmask	Specifies a number of atoms (in ambmask format) that will have their atomic partial charges set to zero. This is mainly for convenience because it removes the need to build additional prmtop files with uncharged atoms for TI calculations involving the removal of partial charges.

21.1.5.1. Input flags specific to Sander

scmask	Specifies the unique (soft core) atoms for this process in ambmask format. This, along with crgmask, is the only parameter that will frequently be different in the two mdin files for V_0 and V_1. It is valid to set scmask to an empty string. A summary of the atoms in scmask is printed at the end of mdout.

21.1.5.2. Input flags specific to PMEMD

scmask1	Specifies the unique (soft core) atoms for V_0 in ambmask format. It is valid to set scmask1 to an empty string.
scmask2	Specifies the unique (soft core) atoms for V_1 in ambmask format. It is valid to set scmask2 to an empty string.

The force field potential energy contributions for the unique atoms in each process will be evaluated separately during the simulation and are recorded after the complete system energy is given:

```
Softcore part of the system:    15 atoms, TEMP(K)    =  316.69
SC_Etot=    24.3248  SC_EKtot=   11.6426  SC_EPtot    =  12.6822
SC_BOND=     4.7723  SC_ANGLE=    2.1411  SC_DIHED    =   1.6096
SC_14NB=     4.2947  SC_14EEL=    0.0000  SC_VDW      =  -0.1355
SC_EEL =     0.0000
SC_RES_DIST= 0.0000  SC_RES_ANG=  0.0000  SC_RES_TORS=   0.0000
SC_RES_PLPT= 0.0000  SC_RES_PLPL= 0.0000  SC_RES_GEN =   0.0000
SC_EEL_DER=  0.0000  SC_VDW_DER=-11.1533  SC_DERIV   = -11.1533
```

The temperatures reported are calculated for the SC atoms only and fluctuate strongly for small numbers of unique atoms. The energies in the first six lines include all terms that involve at least one unique atom, but SC_VDW gives the vdW energy for pairs of unique atoms only which are subject to the standard 12-6 LJ potential. The vdW potential between soft core / non soft core atoms (as given by equation 21.5) is part of the regular VDWAALS term and is counted for dV/dl. The same applies to SC_EEL, which gives only the electrostatic interactions between unique atoms, since electrostatics between soft core / non soft core atoms (for which equation 21.7 is used) are part of regular EEL-energy. Note that the total potential energy, SC_EPtot, does not include contributions from the restraint energies.

SC_EEL_DER, SC_VDW_DER, and SC_DERIV are additional λ-dependent contributions to $\langle \partial V / \partial \lambda \rangle$ that arise from the form of the SC-potentials. For more information on how to perform and setup calculations, please consult the tutorials provided at *http://ambermd.org*.

21.1.5.3. One step transformations using soft core electrostatics

Alternatively to the two-step process of removing charges from atoms first and then changing the vdW parameters of chargeless atoms in a second TI calculation, *sander and pmemd* also have a soft core version of the Coulomb equation implemented for single step transformations under periodic boundary conditions. This is automatically applied to all atoms in scmask and their interactions with common atoms are given by:

$$V_{V_0,disappearing} = (1-\lambda)\frac{q_i q_j}{4\pi\varepsilon_o\sqrt{\beta\lambda + r_{ij}^2}} \tag{21.7}$$

for disappearing atoms. Replace λ by $(1-\lambda)$ and vice versa for the form for appearing atoms. This introduces a new parameter β which controls the 'softness' of the potential. This is set in the input file via:

scbeta	The parameter β in 21.7. Default value is 12A^2, other values have not been extensively tested.

With the use of soft core vdW and electrostatics interactions, arbitrary changes between systems are possible in single TI calculations. However, due to the unusual potential function forms introduced, it is not always clear that a single-step calculation will converge faster than one broken down into several steps. Ref. [419] contains detailed information on the performance of such single step TI calculations.

21.1.6. Preparing TI simulations for use in PMEMD

Since the generation of the *prmtop* file required for *pmemd* TI calculations is slightly more complex, than the generation of two independent prmtop files as required by *sander*, so we provide here a number of examples specific to *pmemd*.

21.1.6.1. Free Energy using linear scaling

For this type of simulation, the molecule is perturbed between the start and end states using linear scaling (Eq. 21.3). This means that V_0 and V_1 must have the same number of atoms. Start by parameterizing the molecule as usual. This may include the addition of dummy atoms as needed. Then, create a pdb which contains both molecules separated by a TER card. Also, update the residue number for the second molecule. If the molecules are different, be sure to use a different residue name for each one. The coordinates for corresponding atoms in the pdb should be the same. The *prmtop* can then be prepared as usual, using *LEaP*. Note that *LEaP* sees both molecules, so it may report a net charge in the prmtop, even though there is no net charge for V_0 or V_1, even after the addition of neutralizing counterions. See Chapter 13 for a complete description of *LEaP*. The input flags for this system are:

```
icfe = 1, timask1 = ':1', timask2 = ':2'
```

Where the first molecule is unique to V_0 and the second molecule is unique to V_1. There may be any number of other molecules, which are treated as common atoms and are part of both V_0 and V_1.

21.1.6.2. Absolute free energy using soft core

For this type of simulation, a molecule is decoupled from the rest of the system using soft core potentials (Eqs. 21.5,21.6). Set up the *prmtop* as you would to run a simulation of the system. The end state is the system with a fully decoupled molecule, so this *prmtop* will also work for TI. The input flags for this system are:

```
icfe = 1, ifsc = 1,
timask1=':1', scmask1=':1',
timask2='', scmask2='',
```

Where the first molecule is the one that is decoupled from the rest of the system at the end state.

21.1.6.3. Relative free energy using soft core

For this type of simulation, a molecule is mutated from one to another using soft core potentials (Eqs. 21.5,21.6). This can be done as a single step using soft core electrostatics (Eq. 21.7), or part of a multistep TI calculation. The *prmtop* is prepared in the same way for both cases. First, parameterize both molecules as usual. Then, create a pdb containing both molecules, separated by a TER card. Additional molecules may be present and will be treated as common atoms. Using this pdb, prepare the system using *LEaP*. The resulting *prmtop* can be used for TI calculations. The input flags for this system are:

```
icfe = 1, ifsc = 1,
timask1=':1', scmask1=':1',
timask2=':2', scmask2=':2',
```

Where the first molecule corresponds to the starting state, and the second molecule corresponds to the ending state. This will set up a single step transformation using soft core electrostatics. To set up a soft core vdW transformation, the flag crgmask=':1 | :2' can be added.

21.1.6.4. Mutation of a protein residue

For this type of simulation, a single residue is mutated in a protein. First, take the pdb for the wildtype and the mutant proteins and concatenate the one after the other, separating them by a TER card. This is necessary because *LEaP* must deal with full molecules. The atoms in the common residues should all have the same coordinates. Any changes to the common residues, such as the addition of disulfide bonds or changing the protonation of HIS, must be done for both copies of the protein in *LEaP*. The output *prmtop* and *inpcrd* files now have two copies of the protein, with one including the mutated residue. Consider a system where residues ':1-5' represent the wildtype protein and residues ':6-10' represent the mutant protein. Furthermore, assume that residue ':3' in the wildtype is mutated, so the corresponding residue in the mutant is residue ':8'. The input flags for this system are:

```
icfe = 1, ifsc = 1,
timask1=':1-5', scmask1=':3',
timask2=':6-10', scmask2=':8',
```

This will do a single step transformation from the wildtype to the mutant protein.

There are a large number of redundant bonding terms that are being calculated, since there are two proteins in the prmtop file. These additional bonding terms can be eliminated, improving the efficiency of the calculation. This is an advanced technique, which is not needed to run a TI simulation, but to have the most efficient calculation. In order to do this, a command has been added to *parmed* to remove these extra terms and atoms as described in Section 14.2.

To run parmed:

```
parmed -p ti.prmtop -i merge.in
```

The input for *parmed* (merge.in) looks like this:

```
loadRestrt ti.inpcrd
setOverwrite True
tiMerge :1-5 :6-10 :3 :8
outparm ti_merged.prmtop ti_merged.inpcrd
quit
```

This will output ti_merged.prmtop and ti_merged.inpcrd which have had redundant bonding terms removed, as well as the masks that should be used in the simulation. The *parmed* output gives:

```
Loaded Amber topology file ti.prmtop
Reading actions from merge.in
Loading restart file ti.inpcrd
Prmtop is overwritable
Merging molecules :1-5 and :6-10 into the same molecule.
Use softcore mask:
timask1='@41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59',
timask2='@77,78,79,80,81,82,83,84,85,86,87,88,89,90',
scmask1='@41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59',
scmask2='@77,78,79,80,81,82,83,84,85,86,87,88,89,90',
Outputting Amber topology file ti_merged.prmtop
Done!
```

Now the input flags for pmemd are:

```
icfe = 1, ifsc = 1,
timask1='@41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59',
timask2='@77,78,79,80,81,82,83,84,85,86,87,88,89,90',
scmask1='@41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59',
scmask2='@77,78,79,80,81,82,83,84,85,86,87,88,89,90',
```

Another possible use is to remove redundant bonding terms in a non soft core simulation. The set up is very similar to that described above, except that the number of atoms in both molecules must be the same. When using *parmed*, ignore the scmask1/scmask2 output, as these are not used in non soft core simulations.

21.1.7. Collecting potential energy differences for FEP calculations

In addition to the Thermodynamic Integration capabilities described above, *sander* can also collect potential energy values during free energy simulation runs for postprocessing by e.g. the Bennett acceptance ratio scheme. This will make sander calculate at given points during the simulation the total potential energy of the system as it would be for different λ-values at this conformation. This functionality is controlled by:

ifmbar	If set to 1 (Default = 0), additional output is generated for later postprocessing.
bar_intervall	Compute potential energies every bar_intervall steps (Default = 100)
bar_l_min	Minimum λ-value (Default = 0.1)
bar_l_max	Maximum λ-value (Default = 0.9)
bar_l_incr	The increment to increase λ by between the minimum and maximum (Default = 0.1)

Such energy collection will normally be part of a regular free energy calculation (using icfe=1 and ifsc=1) involving simulations at various λ-values. Activating this functionality will not have any influence on the simulation trajectory which will evolve according to the preset *clambda* value, it is merely a bookkeeping scheme that removes the necessity of postprocessing output files later.

21.2. Absolute Free Energies using EMIL

As well as comparing two similar systems to find a free energy difference, thermodynamic integration techniques can be used to find the absolute free energy, integrating between an all-atom AMBER model and a simplified model for which the free energy can be directly written down. To find a chemical equilibrium, pairs or sets of absolute free energies must of course be compared to find free energy differences, but taking this “long way around” can be better if the direct integration path between the systems would involve a sharp energetic barrier or a large conformational change. The basic equation of EMIL is thus:

$$A = A_{ref} - \int_0^1 d\lambda \left\langle \frac{\partial \mathscr{H}}{\partial \lambda} \right\rangle_{|\lambda}$$

where A is the total free energy of a system, A_{ref} is the (analytically calculated) free energy of the associated EMIL Hamiltonian and $\mathscr{H}$ is the mixed Hamiltonian, which has the value of the normal AMBER Hamiltonian at $\lambda = 0$ and the EMIL Hamiltonian at $\lambda = 1$.

The method was introduced in the literature with demonstrations for example systems with short-range interactions [420, 421], and an example AMBER calculation for the B-Z conformational equilibrium of DNA also exists [422]. Some further discussion of accuracy and convergence of EMIL calculations using AMBER has also been made [423]. To call EMIL, set *“emil_do_calc” = 1* in the main input file, and also prepare an EMIL-specific input file (by default called “emilParameters.in”).

It is advised to use a Langevin thermostat (*ntt*=3) (section 17.6.7) with a fairly high value of *gamma_ln*, (e.g. 1.0) because dynamics under the EMIL Hamiltonian can have little coupling between particles, therefore an external source of randomness is desirable in order to drive sampling. Use of generalized-Langevin thermostats (section 20.1) is consistent with EMIL, however no study has been made to ascertain the benefits of this approach. Use of SHAKE with EMIL can give unphysical results, so it is advised to turn this off (*ntc = 1, ntf = 1, dt = 0.001*).

The letters “EM” in EMIL refer to “Einstein Molecule”, the name given in the literature [424, 425] to this type of calculation. The use of EMIL is an alternative to other AMBER methods of finding the absolute free energy of molecules in implicit solvent, such as by combining a normal modes analysis (see section 28.10.10) and MM(PB/GB)SA (see chapters 31 and 30). EMIL is quite likely to be more computationally expensive than this type of post-hoc estimate of the free energy carried out after a normal MD simulation, but is also in some ways simpler and is likely to be more accurate in the limit of a large amount of computation being available.

Periodic boundaries can be applied, although EMIL does not support non-rectilinear boxes.

When carrying out an EMIL integration, the AMBER part of the Hamiltonian is gradually turned off with increasing *lambda*. To help achieve this without artifacts, the *emil_sc* option is available (pmemd only) which allows mutual softcoring of all interatomic forces (see section 21.1 , eqn 21.5). For EMIL softcoring there is no need to specify a softcoring mask or modified topology file, as all atoms are included in the process, however *icfe = 1* and *ifsc = 1* must be set if *emil_sc = 1*. The value of *clambda* must also be set, to whatever lambda is also specified in the emil_paramfile. When using *emil_sc* with the default value of *klambda* (=1) (eqn 21.4), there may be sharp changes in the generalized force near to *lambda*=*clambda*=*1*. In this case it is advised to have an integration point at *lambda=clambda=0.99* or a similar value so that the endpoint behaviour is not over-weighted in the total calculation.

When *emil_sc=0* (the default), a less sophisticated approach to the problem of discontinuities in the Lennard-Jones and Coulomb potentials is taken: a short-range repulsion is automatically added to the Hamiltonian for the intermediate stages of the integration *0<lambda<1* in order to prevent atoms from approaching within the problematic regions of the scaled LJ and Coulomb interactions. This method is not always entirely effective, especially in explicit solvent calculations, and may require a cut in the timestep for some values of *lambda*.

EMIL is compatible with multisander and multipmemd (section 17.11), however the only benefit currently is to collect together runs at multiple values of *lambda* for submission as a single job: H-REMD methods (sec 21.5.4) and other advanced uses of multisander and multipmemd have not yet been implemented.

21.2.1. EMIL Namelist Input

An EMIL-specific namelist of input and output filenames for EMIL should be provided in the main input file, of the form:

```
&emil_cntrl
emil_paramfile = "emilParameters.in",
emil_logfile   = "emil.log",
emil_model_infile = "wellsIn.dat",
emil_model_outfile = "wellsOut.dat",
/
```

The variables *emil_paramfile* and *emil_logfile* are paths to files for control data and logging specific to the EMIL calculation. The variable *emil_model_infile* gives the path to an initial specification for an analytically tractable model and *emil_model_outfile* points to a saved model state. If these variables are not set then an initial model will be automatically generated, and no output model will be saved.

21.2.2. EMIL parameter input

The “emilParameters.in” file contains setup info specific to the EMIL calculation. The file is formatted as a list of key-value pairs, one per line. Blank lines or those beginning with a “#” are ignored. The keys are case-insensitive. Providing that you are running at 300K with a fairly standard forcefield, only the *seed, lambda, liquidRes* and *solidRes* values should need to be changed.

The input keys which can be used are:

seed *integer* seed for EMIL’s random number generator

lambda *real* mixing parameter for the alchemical transformation. Must be equal to *sclambda* if *emil_sc=1*.

epsilonWell *real* Depth of harmonic restraints. This is in units of $k_B T$, so that the wells are automatically deeper if the temperature increases. The value of $\beta = 1/k_B T$ at the start of the simulation is printed in the emil_logfile. Harmonic restraints are assigned to atoms of residues in the *solidRes* list and have a potential of the form

$$V(\mathbf{r}) \quad = \quad \varepsilon(r^2/r_{well}^2 - 1).$$

rWell *real* The radius of a harmonic restraint, such that the potential is zero.

epsilonTrap *real* Depth of ’trap’ restraints (in units of $k_B T$). Trap restraints are assigned to atoms of residues in the *liquidRes* lists (if any) and have a potential which is harmonic on $0 \le r \le reqTrap$ and then has a constant force on $reqTrap < r < rTrap$. Beyond *rTrap* the force exerted by a trap well is zero.

reqTrap *real* The radius of the harmonic region of a trap well. Trap wells need to have (at least) a small harmonic region in order to increase the stability of the dynamics near to the bottom of the well.

rTrap *real* The total radius of a trap well.

wingForce *real* The force in the constant-force region of a trap well (in units of $k_B T/\AA$).

solidRes *string* The list of residues for which each atom is permanently assigned to a specific harmonic well.

liquidRes *string* A list of residues which are part of a fluid of chemically identical molecules, for which the chain-well assignment can be adjusted at each timestep by Monte Carlo sampling. Multiple liquids can be defined, in the case that different sets of indistinguishable chains are present in liquid or dissolved phases. Chains whose residues are in these lists are assigned to trap wells, but chains can exchange wells with their neighbours based on a Metropolis acceptance criterion. In each liquid chain only one atom (the heaviest atom is chosen automatically, so this would be the the oxygen of a TIP3P water) interacts directly with the trap well; the remainder of the atoms in the chain have a harmonic well generated for them which holds them in an approximately constant relative position to the 'root' atom of the chain.

swapTriesPerChain *float* Monte Carlo attempt rate for moves that exchange the trap wells between particles in the *liquidRes* lists. The use of swap moves can greatly accelerate convergence, but can also create problems if the acceptance rate (printed in the emil_logfile) is zero or close to zero for any value of lambda. This is equivalent to a freezing-out of the exchange symmetry between particles and requires an adjustment of the reference free energy, as well as probably introducing a discontinuity to the generalized force.

relocTriesPerChain *float* Monte Carlo attempt rate for moves that move particles in the *liquidRes* lists (typically solvent or salt molecules) into or out of their wells. Even if this value is nonzero, relocation moves are only applied if the AMBER Hamiltonian is fully mixed out.

saveWellsEvery *integer* Period with which to write the well positions.

printEvery *integer* Period with which to log the generalized force. The average over the previous non-printed timesteps is output.

Here is an example input file for a fairly standard EMIL run using pmemd and emil_sc:

```
##EMIL input configuration: this is a comment.

##emil has its own RNG
seed            2325

##set the Mixing parameter:
## you will need several values on the interval [0,1].
lambda          0.0

##Residue names associated with wells.
##This is the list of residues needed for duplex DNA
##you will have to extend/change it for your own system
solidRes   DC,DG,DA,DT,DA5,DT5,DA3,DT3,DG5,DG3,DC5,DC3
liquidRes WAT
liquidRes NA
liquidRes CL

swapTriesPerChain 0.1

##timesteps between writing well positions
saveWellsEvery 100000

##timesteps between output of generalized force
printEvery      1000
```

21.2.3. EMIL generalized-force output

EMIL writes its output to the emil_logfile. This logfile contains some header information, and data to monitor the progress of the run, but the important lines are of the following format:

```
nstep:  25 soft_dHdL:  2.06419354e+04 molec_dHdL:...
   ...6.13140526e+04 abstr_dHdL:  -5.34856062e+02
```

The step number, *nstep*, indicates the timestep at which the printout was made. The *soft_dHdL* is the generalized force due to the weak and short-range repulsive term which is present in the mixed Hamiltonian for values of $0 < \lambda < 1$, but only if *emil_sc=0*. The *molec_dHdL* is the generalized force due to the AMBER Hamiltonian, and the *abstr_dHdL* is the generalized force due to the EMIL Hamiltonian. The gradient of the total Hamiltonian with respect to *lambda* is just the sum of these three terms. In order to make the most efficient use of information, EMIL accumulates a mean value of each generalized force term between printouts, so the value written is not an instantaneous "snapshot" but the average over a time window *printEvery* steps in length.

Although the EMIL Hamiltonian is specified in units of $k_B T$, the generalized force is output in units kcal/mol, so the strength of the restraints (and the size of the generalized force) will increase with temperature.

21.2.4. EMIL tractable model definition

The model defined by EMIL is currently very simple. Each atom of any residues in the list *solidRes* from "emilParameters.in" is restrained to a fixed position using a harmonic well of depth *epsilonSolid*, with the zero of the potential at distance *rWellSolid*. The position of the harmonic well minimum is fixed at whatever the atom position at the start of the run might be, unless the option *readStartWellFileName* is provided, in which case the positions are read in from the file.

Atoms defined by the *liquidRes* lists have wells with a finite range, and in order to have faster convergence for simulations including explicit solvent (where the particle-well distance can otheriwise be very large at small λ) the particle-well assignment is shuffled at each timestep by Monte Carlo sampling. The MC method is not currently implemented in parallel, which can create limitations for EMIL calculations using large numbers of cores per value of λ: the optimal parallelisation strategy in this case is to make many runs on few cores each, at different values of λ.

Derivations and formulae for the free energy associated with each well type are available in the supplementary data of [422], however the calculated totals are also printed out at the start of the *emil_logfile*.

Use of thermostat synchronisation to reduce errorbars

A feature of the Langevin thermostat which can cause serious problems in other circumstances (discussed in [352]) is that simulations run with the same seed will come to resemble each other, even if the Hamiltonians and initial configurations are somewhat different. A surprising benefit of this is that, if EMIL is used to compare two or more dissimilar systems then the variance of the difference in the generalized forces at a given value of λ can be less than the sum of the variances of the individual measurements:

$$VAR\left[\frac{\partial\mathcal{H}(x_1,\lambda)}{\partial\lambda} - \frac{\partial\mathcal{H}(x_2,\lambda)}{\partial\lambda}\right] < VAR\left[\frac{\partial\mathcal{H}(x_1,\lambda)}{\partial\lambda}\right] + VAR\left[\frac{\partial\mathcal{H}(x_2,\lambda)}{\partial\lambda}\right] \tag{21.8}$$

which is to say that, although the means of the two generalized forces are estimated correctly, the covariance of the two generalized forces is greater than zero. Using this phenomenon it is possible to estimate the difference in free energies between two (or N) systems more cheaply than the free energies themselves [422, 423, 426].

While it is therefore beneficial to use the same seeds for a given value of λacross all systems, it is still necessary to use a new seed for each restart of the same trajectory, and to use different seeds for different values of λ. To maintain thermostat synchronization, the number of atoms in the different systems must be the same. This can be achieved if necessary by the addition of non-interacting dummy atoms to the smaller topology files using the *parmed* (sec. 14.2) utility.

Brief instructions for an EMIL calculation

To run an EMIL calculation, first equilibrate a single simulation of the system in question then follow the steps below:

1. If you started off at constant pressure, find the average box-size and scale the system to this size.
2. Prepare multiple "emilParameters.in" files (see section 21.2.2) which differ from each other only in the parameters *seed* and *lambda*. The values of lambda should be spread over the interval $0 \leq \lambda \leq 1$.
3. Put your "emilParameters.in" files into one directory each and run *pmemd* in each of the directories, setting *ntt = 3, ntp = 0, emil_do_calc=1, emil_sc=1.* If runs finish and are restarted, then the saved well positions written at the end of the old run will need to be loaded into the new one, as well as the normal AMBER restart files.
4. It may be necessary to set up restraints of some kind from within *pmemd* or *sander* if the free energy to be calculated is for only a subset of the available conformations of the molecule(s), or to speed up convergence at low values of λ, by preventing the solute molecule from drifting away from its restraint system (this drift is a particular problem for small systems, where the cumulative effect of the EMIL solute restraints, even over all atoms, is still weak at small λ).
5. Collect the converged time-average values of the generalized forces (or the differences in generalized forces if you are comparing several systems) at each value of λ. It is often worth looking at the different time series individually, in order to make the most efficient use of data by only throwing away the minimum number of equilibration points, and in order to target simulation effort to those values of λ which are taking the longest to give a small errorbar [422]).
6. Do a numerical integration of each of the three *dHdL* terms from the EMIL logfiles with respect to λ then subtract these totals from the free energy of the EMIL Hamiltonian, which is printed in the headers of the EMIL logfiles, to get the free energy of the system under the AMBER Hamiltonian. As well as taking time-averages of the (delta) generalized forces and then integrating these values, it may also be valuable to collect a time-series of the (delta) free energy values and examine this total for convergence.

A longer tutorial on the use of EMIL is available on the AMBER website, also the examples in the test suite might provide some help to get started.

21.3. Linear Interaction Energies

sander contains rudimentary facilities to compute binding free energies using the linear interaction energy model.

ilrt	if set to 1, turns on the computation of LIE contributions (default=0)
lrt_interval	Computer LIE contributions every lrt_interval MD steps (default=50)
lrtmask	The 'solute'. Interaction Energies between the atoms in lrtmask and the remainder of the system are computed.

The LIE facilities work by computing the system energies several times using different charge and vdW-parameter sets. This results in reduced performance if lrt_interval is set to less than approx. 10. The LIE output at the end of the *mdout* file gives the electrostatic interaction energy between the solute and rest of the system *times 0.5*, i.e. in accordance with the original formulation of LIE theory. The solute SASA and vdW-interaction energy with its surroundings is calculated unscaled.

21.4. Umbrella sampling

Another free energy quantity that is accessible within *sander* is the ability to compute potentials of mean force (at least for simple distance, angle, or torsion variables) using umbrella sampling. The basic idea is as follows. You add an artificial restraint to the system to bias it to sample some coordinate in a certain range of values, and you keep track of the distribution of values of this coordinate during the simulation. Then, you repeatedly move the minimum of the biasing potential to different ranges of the coordinate of interest, and carry out more simulations. These different simulations (often called "windows") must have some overlap; that is, any particular value of the coordinate must be sampled to a significant extent in more than one window. After the fact, you can remove the effect of the bias sing potential, and construct a potential of mean force, which is the free energy profile along the chosen coordinate.

The basic ideas have been presented in many places,[252–254, 427, 428] and will not be repeated here. The implementation in *sander* follows two main steps. First, restraints are set up (using the distance and angle restraint files) and the DUMPFREQ parameter is used to create "history" files that contain sampled values of the restraint coordinate. Second, a collection of these history files is analyzed (using the so-called "weighted histogram" or WHAM method [252–254]) to generate the potentials of mean force. As with thermodynamic integration, the *sander* program itself does not compute these free energies; it is up to the user to combine the output of several windows into a final result. For many problems, the programs prepared by Alan Grossfield (http://membrane.urmc.rochester.edu/) are very convenient, and the *sander* output files are compatible with these codes. Other methods of analysis, besides WHAM, may also be used.[429]

A simple example. The input below shows how one window of a potential of mean force might be carried out. The coordinate of interest here is the chi angle of a base in an RNA duplex. Here is the *mdin* file:

```
test of umbrella sampling of a chi torsion angle
&cntrl
nstlim=50000, cut=20.0, igb=1, saltcon=0.1,
ntpr=1, ntwr=50000, ntt=3, gamma_ln=0.2,
ntx=5, irest=1,
ntc=2, ntf=2, tol=0.000001,
dt=0.001, ntb=0,
nmropt=1,
/
&wt type='DUMPFREQ', istep1=10 /
&wt type='END' /
DISANG=chi.RST
DUMPAVE=chi_vs_t.170
```

The items in the *&cntrl* namelist are pretty standard, and not important here, except for specifying *nmropt=1*, which allows restraints to be defined. (The name of this variable is an historical artifact: distance and angle restraints were originally introduced to allow NMR-related structure calculations to be carried out. But they are also very useful for cases, like this one, that have nothing to do with NMR. Please see section 23.1 for a complete description of the options, noting especially the *restraint* and *rstwt* variables.) The DUMPFREQ command is used to request a separate file be created to hold values of the torsion angle; this will have the name *chi_vs_t.170* given in the DUMPAVE file redirection command.

The torsion angle restraint itself is given in the *chi.RST* file:

```
# torsion restraint for chi of residue 2
&rst iat=39,40,42,43, r1=0., r2=170., r3=170., r4=360., rk2 = 30.,
rk3 = 30., /
```

The *iat* variable gives the atom numbers of the four atoms that define the torsion of interest. We set $r2 = r3$ and *rk2* = *rk3* to obtain a harmonic bias sing potential, with a minimum at 170 o . The values $r1$ and $r4$ should be far away from 170, so that the potential is essentially harmonic everywhere. (It is not required that bias sing potentials be harmonic, but Dr. Grossfield's programs assume that they are, so we enforce that here.) Subsequent runs would change the minimum in the potential to values other than 170, creating other *chi_vs_t* files. These files would then be used to create potentials of mean force. Note that the conventionally defined "force constant" is twice the value *rk2*, and that the Grossfield program uses force constants measured in degrees, rather than radians. So you must perform a unit conversion in using those programs, multiplying *rk2* by 0.0006092 ($= 2(\pi/180)^2$) to get a equivalent force constant for a torsional restraint.

21.5. Replica Exchange Molecular Dynamics (REMD)

Replica exchange molecular dynamics (REMD) is an *expanded ensemble* method—it samples from an ensemble (significantly) larger than a typical statistical mechanical ensemble defined by the Hamiltonian governing the system (e.g., microcanonical, canonical, grand-canonical, etc.). This section will briefly describe the general theory of REMD-based techniques, after which the later subsections will cover the details of Amber's REMD implementation as well as the various allowable exchange types.

21.5.1. Introduction

'Sampling' in expanded ensemble techniques can be broadly decomposed into two different types of sub-sampling within the total ensemble. The first type is the common conformational sampling that can be realized through methods like molecular dynamics. The second type samples from thermodynamic state-space, in which the core Hamiltonian (together with its thermodynamic constraints) that defines the ensemble is allowed to change. Thus, expanded ensemble techniques broaden the sampling space of simulations by allowing a system to move through both conformational space (which is typically continuous) and state space (which is typically discrete). A point in the phase space of the expanded ensemble is defined by a specific value(s) of the state parameter(s) in addition to the $3N$ particle positions and conjugate momenta. To simplify terminology, I will refer to all of the points in phase space that have the same value of the state parameter(s) as a 'sub-ensemble' since it can be interpreted as the statistical ensemble defined by that thermodynamic state, whereas 'the ensemble' will refer to the full, expanded ensemble containing all state indices.

To ensure that the ensemble is constructed properly, the simulation must generate a reversible Markov chain of states. Standard MD obeys this requirement under the (ubiquitous) assumption that the system is ergodic. For Monte Carlo-based methods, a reversible Markov chain implies that the condition of detailed balance (Eq. 21.9) is satisfied. Detailed balance is effectively an equilibrium condition, in which the probability of being in state i (π_i) multiplied by the transition probability of moving from state i to j ($P_{i\rightarrow j}$) is equal to the probability of being in state j multiplied by the transition probability of moving from state j to i. By relating probabilities with concentrations and transition probabilities with chemical rate constants, it is easy to see that Eq. 21.9 is a simple equilibrium equation between two species.

$$\pi_i P_{i\rightarrow j} = \pi_j P_{j\rightarrow i} \tag{21.9}$$

To sample from the ensemble, REMD employs a set of non-interacting replicas (i.e., the forces on the particles are unaffected by the particles in other replicas) that attempt to swap their positions in state space during the course of the simulation. In Amber, the conformational sampling in each replica (i.e., sub-ensemble) is peformed with MD while replica swaps in state space are performed using Metropolis Monte Carlo in which the exchange probability is computed from Eq. 21.9.

The general workflow used by Amber for replica exchange simulations is illustrated in Figure 21.1. Each replica runs a pre-specified number of MD steps before stopping to attempt exchanges with one of its nearest neighbor (its exchange partner alternates every other exchange attempt). Restricting exchange attempts to pairs is not required (exchanges can involve 3 replicas, for instance), but it greatly simplifies the resulting exchange probability when solving Eq. 21.9, and allows $N/2$ consecutive exchanges to be attempted independently. Furthermore, replica exchange attempts need not be made deterministically or synchronously (i.e., each replica evaluates exchange attempts at the same time relative to each other), but doing so significantly simplifies the programming requirements. The following sections will describe how REMD is implemented and performed in Amber for each of the supported types of exchanges—temperature, solution pH, and generalized Hamiltonian.

If you are not already familiar with the technique and its theoretical underpinnings, we recommend that you study the literature, particularly of the type of replica exchange you plan on using.[430–435]

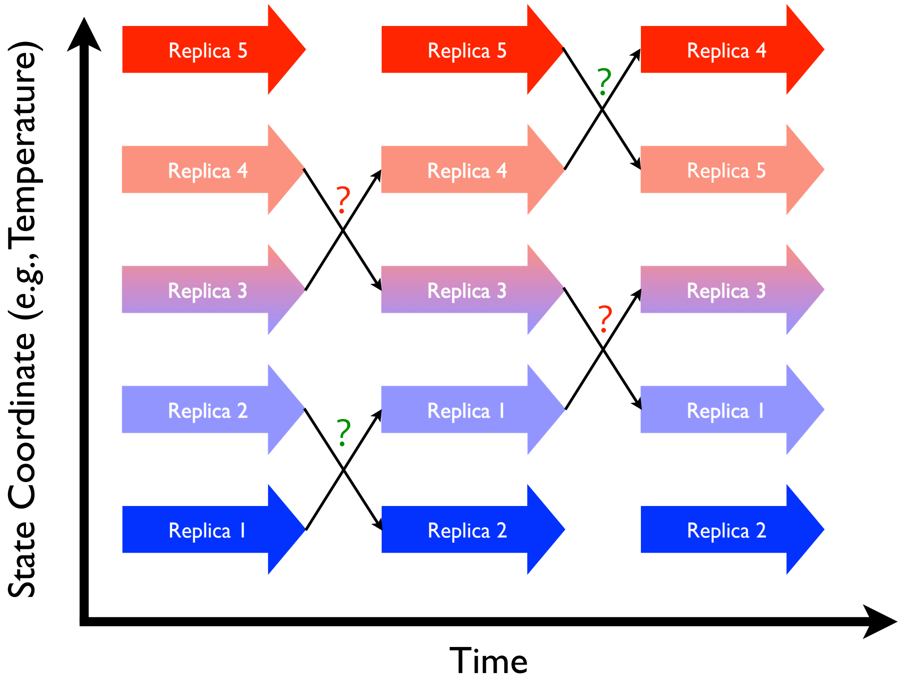

State Coordinate (e.g., Temperature)
Time
Replica 5
Replica 4
Replica 3
Replica 2
Replica 1
?
?
Replica 5
Replica 4
Replica 3
Replica 1
Replica 2
?
?
Replica 4
Replica 5
Replica 3
Replica 1
Replica 2

Figure 21.1.: *Replica exchange schematic showing 5 replicas combined in an expanded ensemble. Large arrows represent MD trajectories of sub-ensembles while the smaller arrows represent attempted swaps between replicas in state space. Question marks represent Monte Carlo exchange attempts (green for successful, red for failed).*

21.5.2. Running REMD simulations

In order to run REMD simulations, *sander* and *pmemd* use the *multisander* (and *multipmemd*) machinery that allows multiple MD trajectories to be run in the same simulation. This mode of sander and pmemd is used slightly differently than their normal operation, and is described in Section 17.11.

There are two new variables that must be present in the `&cntrl` section of the mdin files of each replica for all REMD simulations.

numexchg This is the number of exchange attempts that will be performed between replica pairs

nstlim This is the number of MD steps that will be performed between exchange attempts

There are also additional command-line flags that should be placed on the command-line (with the groupfile), described below:[1]

-rem <#> This flag defines the type of replica exchange that will be run for 1-D REMD. The allowed values are 1 (T-REMD), 3 (H-REMD), and 4 (pH-REMD). Each approach is described in later sections.

-remlog <remlog_file> This flag specifies the name of the log file that contains information about each of the replicas during each exchange attempt. The default value is `rem.log`.

-remtype <remtype_file> This flag specifies a filename for the remtype file; this file provides helpful information about the current replica run. For reservoir REMD runs it also prints reservoir information. Default is `rem.type`

Some specialized types of REMD simulations have additional command-line flags that will be described in future sections.

Note the change in the meaning of `nstlim`. For standard MD, `nstlim` is the total number of MD steps that will be performed. For REMD, simulations, on the other hand, `nstlim` is the number of steps between replica exchange attempts and the total number of steps is equal to $nstlim \times numexchg$. The `nstlim` variable, then, is related to the inverse of the exchange attempt frequency (EAF) in REMD simulations. The value of `numexchg` ***must*** be the same for each input file, or the program will hang indefinitely as those replicas assigned more exchange attempts wait to exchange data with replicas that have already finished. We also strongly suggest keeping `nstlim` the same as well to avoid making replicas with fewer steps wait for those with more steps to finish.

Currently, Amber only supports running replica exchange simulations at constant volume for explicit solvent simulations, and all replicas must have the same volume. Therefore, the equilibration stage of each replica should begin *after* the original system was run at constant pressure to stabilize the density (and volume).

Amber currently supports 4 types of exchange attempts: Temperature REMD (T-REMD), Hamiltonian REMD (H-REMD), constant pH REMD (pH-REMD), and replica exchange self-guided Langevin dynamics (RXSGLD). Multi-dimensional REMD simulations (defined by 2 or more state parameters) are also supported. The instructions given above apply to all REMD simulations, but instructions for running REMD simulations in general strongly depend on the type of exchange being attempted. Additional details for running REMD simulations are provided in the following sections for each type of exchange attempt.

21.5.3. Running Temperature REMD simulations

In temperature REMD (T-REMD), replicas are distinguished based on the temperature of their temperature bath. In general, each replica should differ from each other *only* by their target temperature, `temp0`, specified in the mdin file for each replica. The N replicas are first sorted in an array by their target temperatures, so the ordering of the replicas in the groupfile is irrelevant. Neighboring residues attempt to exchange every `nstlim` MD steps, with the exchange partners alternating each replica exchange attempt. For example, if replicas 2 and 3 attempt to swap the first time then replicas 1 and 2 will attempt to swap the next time (as will replicas 3 and 4). Topologically, the N temperature-sorted replicas form a loop, in which the first and the last replicas are neighbors. Therefore, $N/2$ exchanges are attempted every `nstlim` steps. The exchange success rate is computed via a Metropolis Monte

---

[1] Some specialized types of REMD simulations have additional command-line flags that will be described in the later sections.

Carlo move shown in Eq. 21.10 that satisfies detailed balance for swapping temperatures. If the exchange is allowed between the pair, the temperature between the replicas is swapped before MD resumes. The velocities of each replica involved in successful exchange are then adjusted by the scaling factor $\sqrt{T_{new}/T_{old}}$ where T_{old} is the temperature before the exchange and T_{new} is the temperature after. This velocity scaling is done to ensure that each structure is immediately adjusted to its new target temperature. After the exchange calculation, the MD resumes for `nstlim` steps until the next exchange attempt (in which the exchange partners alternated with respect to the previous exchange attempt).

$$P_{i,j} = \min\left\{1, \exp\left[-\left(\beta_i - \beta_j\right)\left(E_j - E_i\right)\right]\right\} \tag{21.10}$$

Before starting a replica exchange simulation, an optimal set of temperatures should be determined so that the exchange ratio is roughly a constant. This spacing of the replicas in temperature-space determines the probability of exchange among the replicas, and the user is referred to the literature for a more complete description of the influence of various factors on the exchange probability. A useful resource for generating a series of temperatures with a specific exchange success probability can be found online at http://folding.bmc.uu.se/remd/.

Each replica requires (for input files) or generates (for output files) its own *mdin, inpcrd, mdout, mdcrd, restrt, mdinfo,* and associated files. The names are provided through the specification of a *groupfile* on the command line with the *-groupfile groupfile* option. The *groupfile* file contains a separate command line for each of the replicas or multisander instances, as described in Section 17.11. To choose the number of replicas or multisander instances, the *-ng N* command line option is used (in this case to specify *N* separate instances.)

For example, an 4-replica REMD job will need 4 mdin and 4 inpcrd files. Then, the groupfile might look like this:

```
#
# multisander or replica exchange group file
#
-O -i mdin.rep1 -o mdout.rep1 -c inpcrd.rep1 -r restrt.rep1 -x mdcrd.rep1
-O -i mdin.rep2 -o mdout.rep2 -c inpcrd.rep2 -r restrt.rep2 -x mdcrd.rep2
-O -i mdin.rep3 -o mdout.rep3 -c inpcrd.rep3 -r restrt.rep3 -x mdcrd.rep3
-O -i mdin.rep4 -o mdout.rep4 -c inpcrd.rep4 -r restrt.rep4 -x mdcrd.rep4
```

Note that for T-REMD the *mdin* and *inpcrd* files are *not* required to be ordered by their target temperatures since they will not remain sorted during the simulation. Sorting is performed automatically at each REMD iteration as described above. Thus one can restart REMD simulations without modifying the restart files from the previous REMD run (see below for more information about restarting REMD).

It is important when running T-REMD to ensure that each topology file is equivalent and the input files differ only in the temperature (`temp0`), and that all explicit solvent calculations are run at constant volume. Because Eq. 21.10 was derived under the assumption that exchanging replicas only swapped temperatures, only the temperature can vary between replicas. Satisfying this requirement is left to the user, and no warnings or checks are implemented if this assumption is violated.

21.5.3.1. Restarting REMD simulations

It is recommended that each REMD run generate a new set of output files (such as mdcrd), but for convenience one may use -A in the command line in order to append output to existing output files. This can be a useful option when restarting REMD simulations. If -A is used, files that were present before starting the REMD simulation are appended to throughout the new simulation. If -O is used, any files present are overwritten. The recommended input file settings for restarting a REMD simulation are ig=-1 to use the wall clock for the pseudo-random number generator seed, ntxo=2 to write a NetCDF restart file, ioutfm=1 to write NetCDF trajectories, irest=1 to restart, and ntx=5 to read velocities; the first three should be used in the initial calculation.

At the end of a REMD simulation, the target temperature of each replica is most likely not the same as it was at the start of the simulation (due to successful exchanges). If one wishes to continue this simulation, *sander* or *pmemd* will need to know how the target temperature has changed. Since the target value is normally specified in the mdin file (via `temp0`), the previous mdin files would all need to be modified to reflect changes in target temperature of each replica. In order to simplify this process, *sander* and *pmemd* write the final target temperature as additional information in the restart files during a T-REMD simulation. When a T-REMD simulation is started, the program will check to see if the target temperature is present in the restart file. If it is present, this value will override the value in the mdin file. In this manner, one can restart the simulation from the set of restart files and *sander* or *pmemd* will automatically update the target temperature of each replica to correspond to the final value from the previous run. If the target temperature is not present (as would be the case for the first REMD run), the correct values must be present in the mdin files.

21.5.3.2. Content of the output files

It is important to note that in the current implementation of T-REMD *all output is* ***by replica only, not by temperature***! To facilitate post processing of trajectory data by temperature, the temperature must be specified for each snapshot. For NetCDF trajectories, adding this information is simple because NetCDF is an extensible format. We strongly recommend that you always use NetCDF trajectories, especially for REMD simulations. For ASCII formatted trajectories, a header line is written to each frame just before the coordinates. This header line has the format:

```
REMD <replica#> <exchange#> <step#> <Temperature>
```

PTRAJ and CPPTRAJ are be able to read trajectories with this new format.

The rem.log file for T-REMD simulations has the following format:

```
# Replica Exchange log file
# numexchg is         5
# REMD filenames:
#   remlog= rem.log
#   remtype= rem.type
# Rep#, Velocity Scaling, T, Eptot, Temp0, NewTemp0, Success rate (i,i+1), ResStruct#
# exchange       1
 1      1.15      0.00    -10.46    300.00    400.00      0.00      -1
 2      1.04      0.00    -10.46    325.00    350.00      2.00      -1
 3      0.96      0.00    -10.46    350.00    325.00      0.00      -1
 4      0.87      0.00    -10.46    400.00    300.00      2.00      -1
# exchange       2
 1      0.94    312.03     -6.81    400.00    350.00      1.00      -1
 2      1.07    280.77     -3.95    350.00    400.00      1.00      -1
 3     -1.00    247.11    -10.58    325.00    325.00      1.00      -1
 4     -1.00    271.12    -14.15    300.00    300.00      0.00      -1
# exchange       3
 1      0.96    305.31    -11.02    350.00    325.00      0.67      -1
 2      0.87    288.89    -12.45    400.00    300.00      1.33      -1
 3      1.04    290.99    -13.30    325.00    350.00      1.33      -1
 4      1.15    256.19    -12.83    300.00    400.00      0.00      -1
```

The columns, listed in order, are the replica number, velocity scaling factor, the instantaneous temperature, the potential energy of the structure, the target temperature before the exchange attempt, the target temperature after the exchange attempt, the average success rate, and the reservoir structure number. The replica number never changes since replicas swap target temperatures. When the velocity scaling factor is -1, the exchange attempt failed and velocities are not altered. For successful exchange attempts, velocites are either scaled up (when the new target temperature is higher than the old one) or down (when the new target temperature is lower than the old one). Success rates are calculated as $\#successes/\#tries \times 2$, where the factor of 2 is used because each pair of neighboring replicas attempts to exchange every other exchange attempt. In the beginning of the log file, this may lead to unusual success rates, but after a large number of exchange attempts the values will normalize. Success rates are computed as the exchange success rate between the original temperature (Temp0 column) and the next highest temperature in the temperature ladder (not necessarily the temperature it just attempted to exchange with). The success rate for the highest temperature is often 0 since it reflects the success rate between the highest and lowest temperatures.

All temperatures are reported in Kelvin and all energies in kcal/mol.

21.5.3.3. Cautions when using replica exchange

While many variations of replica exchange have been tested with sander, all possible variations have not been tested and the option is intended for use by advanced researchers that already have a comprehensive understanding of standard molecular dynamics simulations. Caution should be used when creating REMD input files. Amber will check for the most obvious errors but due to the nature of the multiple output files the reason for the error may not be readily apparent. The following is only a subset of things that users should keep in mind:

1. The number of replicas must be an even number (so that all replicas have a partner for exchange).
2. Temp0 values for each replica must be unique for Temperature-based REMD.
3. Other than temp0 mdin files should be identical.
4. Temp0 values should not be changed in the nmropt=1 weight change section.
5. A groupfile is required.
6. If high temperatures are used, it may be necessary to use a smaller time step and possibly restraints to prevent cis/trans isomerization or chirality inversion.
7. Due to increased diffusion rates at high temperature, it may be good to use iwrap=1 to prevent coordinates from becoming too large to fit in the restart format. An alternative to this is, of course, to use the default NetCDF restart files (ntxo=2).
8. Note that the optimal temperature range and spacing will depend on the system. The user is strongly recommended to read the literature in this area.
9. Constant pressure is not supported for REMD simulations. This means NTP must be 0.

21.5.3.4. Replica exchange example

Below is an example of an 8-replica REMD run on 16 processors, (note that launching a MPI program varies from computer to computer).

```
mpirun -np 16 sander.MPI -ng 8 -groupfile groupfile -rem 1
```

Here is the groupfile:

```
#
# multisander or replica exchange group file
#
-O -i mdin.rep1 -o mdout.rep1 -c inpcrd.rep1 -r restrt.rep1 -x mdcrd.rep1
-O -i mdin.rep2 -o mdout.rep2 -c inpcrd.rep2 -r restrt.rep2 -x mdcrd.rep2
-O -i mdin.rep3 -o mdout.rep3 -c inpcrd.rep3 -r restrt.rep3 -x mdcrd.rep3
-O -i mdin.rep4 -o mdout.rep4 -c inpcrd.rep4 -r restrt.rep4 -x mdcrd.rep4
-O -i mdin.rep5 -o mdout.rep5 -c inpcrd.rep5 -r restrt.rep5 -x mdcrd.rep5
-O -i mdin.rep6 -o mdout.rep6 -c inpcrd.rep6 -r restrt.rep6 -x mdcrd.rep6
-O -i mdin.rep7 -o mdout.rep7 -c inpcrd.rep7 -r restrt.rep7 -x mdcrd.rep7
-O -i mdin.rep8 -o mdout.rep8 -c inpcrd.rep8 -r restrt.rep8 -x mdcrd.rep8
```

This input specifies that T-REMD should be used (-rem 1), with 8 replicas (-ng 8) and 2 processors per replica (-np 16). Note that the total number of processors should always be a multiple of the number of replicas.

21.5.3.5. Replica exchange using a hybrid solvent model

This section describes an advanced feature of Amber.[146, 147] Users that are not already comfortable with standard replica exchange simulations should likely get more experience with them before attempting hybrid solvent REMD calculations.

For large systems, REMD becomes intractable since the number of replicas needed to span a given temperature range increases roughly with the square root of the number of degrees of freedom in the system. Recognizing that the main difficulty in applying REMD with explicit solvent lies in the number of simulations required, rather than just the complexity of each simulation, we recently developed a new approach in which each replica is simulated in explicit solvent using standard methods such as periodic boundary conditions and inclusion of long-range electrostatic interactions using PME. However, the calculation of exchange probabilities (which determines the temperature spacing and thus the number of replicas) is handled differently. Only a subset of closest water molecules is retained, with the remainder temporarily replaced by a continuum representation. The energy is calculated using the hybrid model, and the exchange probability is determined. The original solvent coordinates are then restored and the simulation proceeds as a continuous trajectory with fully explicit solvation. This way the perceived system size for evaluation of exchange probability is dramatically reduced and fewer replicas are needed.

An important difference from existing hybrid solvent models is that the system is fully solvated throughout the entire MD simulation, and thus the distribution functions and solvent properties should not be affected by the use of the hybrid model in the exchange calculation. In addition, no restraints of any type are needed for the solvent, and the solute shape and volume may change since the solvation shells are generated for each replica on the fly at every exchange calculation. Nearly no computational overhead is involved since the calculation is performed infrequently as compared to the normal force evaluations. Thus the hybrid REMD approach can employ more accurate continuum models that are too computationally demanding for use in each time step of a standard molecular dynamics simulation. However, since the Hamiltonian used for the exchange differs from that employed during dynamics, these simulations are approximate and are not guaranteed to provide correct canonical ensembles.

At each exchange calculation sander will create the hybrid system based on the current coordinates for the fully solvated system. This is done by calculating the distance of each water oxygen to the nearest solute atom, and sorting the water by increasing shortest distance. The closest *numwatkeep* are retained and the potential energy is calculated using the GB model specified by *hybridgb*. After the energy calculation the fully solvated system is restored.

For a more complete example, users are directed to the hybridREMD test case (in the rem_hybrid subdirectory) in the Amber test directory.

numwatkeep The number of explicit waters that should be retained for the calculation of potential energy to be used for the exchange calculation. Before each exchange attempt, the closest numwatkeep waters will be retained (closest to the solute) and the rest will be temporarily removed and then replaced after the exchange probability has been calculated. The default value is -1, indicating that all waters should be retained (standard REMD). A value of 0 would direct Amber to remove all of the explicit water (as in MM-PBSA) while a nonzero value will result in some water close to the solute being retained while the rest is removed. Currently it is not possible to select a subset of solute atoms for determining which waters are "close". Determining the optimal *numwatkeep* value is a topic of current research.

hybridgb Specifies which GB model should be used for calculating the PE of the stripped coordinates, equivalent to the *igb* variable. Currently only *hybridgb* values of 1, 2, and 5 are supported.

Cautions: Hybrid-REMD has not been extensively tested. The following would not be expected to work without further modification of the code:

1. Only the water is imaged for the creation of the stripped system. Care should be taken with dimers (such as DNA duplexes) to ensure that the imaging is correct.
2. Explicit counterions should probably not be used.
3. The choice of implicit solvent model will likely have a large effect on the resulting ensemble.

21.5.3.6. Reservoir REMD

The ability to perform REMD with a structure reservoir [436, 437] has been implemented in Amber as of version 10. Although REMD can significantly increase the efficiency of conformational sampling, obtaining converged data can still be challenging. This is particularly true for larger systems, as the number of replicas needed to span a given temperature range increases with the square root of the number of degrees of freedom in the system. Another consideration is that the folding rate of a peptide tends not to be as dependent on temperature as the unfolding rate, making the search for native peptide structures in higher temperature replicas more problematic; in the case where a native-like structure is found it will almost always be exchanged to a lower temperature replica, requiring a repeat of the search process. In addition, the exchange criterion in REMD assumes a Boltzmann-weighted ensemble of structures, which is typically not the case at the start of a REMD simulation. Although the exchange criterion will eventually drive each replica toward a Boltzmann-weighted ensemble of structures, this essentially means that until all of the replicas are converged, none of the replicas are converged.

Reservoir REMD is a method which can significantly enhance the rate of convergence and reduce the high computational expense of standard REMD simulations. An ensemble of structures (or reservoir) is generated at high temperature, then linked to lower temperatures via REMD. Periodic exchanges are attempted between randomly chosen structures in the reservoir and the highest temperature replica. If the structure reservoir is already Boltzmann-weighted,[436] convergence is significantly enhanced as the lower temperature replicas simply act to re-weight the reservoir ensemble - in essence all of the searching has been accomplished from the start. This is in contrast to standard REMD where all the replicas are run simultaneously, and the computational expense for running long simulations must be paid for each of the replicas even though only a few high-temperature ones may be contributing to sampling of new basins.

One major advantage of this approach is that a converged ensemble of conformations needs to be generated only once and only for one temperature. Typically this temperature should be high enough to facilitate crossing of energy barriers, but low enough that there is still a measureable fraction of native structure present. Another advantage is that exchanges with the reservoir do not need to be time-correlated with the replica simulations; folding events sampled during reservoir generation can provide multiple native structures for the other replicas.

It may not always be possible however to generate a Boltzmann-weighted ensemble of structures (e.g. for a large molecule in explicit solvent). In such cases it is possible to use a non-Boltzmann weighted reservoir by modifying only the exchange criterion between the reservoir and the highest temperature replica. If the weight of all structures in the reservoir is set to 1, this corresponds to a completely flat distribution across the free energy landscape. Alternatively, weights can be assigned to structures based on various structural properties. In the current implementation, weights are assigned to structures via dihedral bin clustering, wherein clusters are identified by unique configurations of user-defined dihedral angles.

There are several new command line arguments that pertain to Reservoir REMD:

-rremd Type of reservoir to use.

- **= 0** No reservoir (Default)
- **= 1** Boltzmann-weighted reservoir
- **= 2** Non-Boltzmann weighted reservoir where the weight of each structure in the reservoir is assumed to be 1/N
- **= 3** Non-Boltzmann weighted reservoir with weights defined by dihedral angle binning.

-reservoir Specifies the file name prefix for reservoir structures. Reservoir structure files should be in the restart file format *MDRESTRT*, and are expected to be named according to the format <name>.XXXXXX, where XXXXXX is a 6 digit integer, e.g. frame.000001. Default is "reserv/frame". **IMPORTANT NOTE:** Structure numbering should begin at 1.

-saveene specifies the file containing energies of the structures in the reservoir (default filename is "saveene"). This file must contain a header line with format:

```
<# reservoir structures> <reservoir T> <#atoms> <random seed>
<velocity flag>
```

If the velocity flag =1 then velocity information will be read from the reservoir structure files, otherwise (if velocity flag =0) velocities will be assigned to the structure based on the reservoir temperature. After the header line there should be a line containing the potential energy of each reservoir structure. **IMPORTANT NOTE:** For reservoir REMD with dihedral bin clustering (rremd==3) each potential energy should be followed by the cluster # that reservoir structure belongs to.

-clusterinfo For reservoir REMD with dihedral bin clustering (rremd==3) this file specifies what dihedrals are used and the binsize, as well as what cluster each reservoir structure belongs to. Default is "cluster.info". File has the following format:

```
<# Dihedral Angles>
<atom# 1>  <atom# 2>  <atom# 3>  <atom# 4> [Dihedral 1]
. .
. .
. .
<atom# 1>  <atom# 2>  <atom# 3>  <atom# 4> [# Dihedral Angles]
<Total # Clusters>
<Cluster #>  <Weight>
<Bin1><Bin2>...<Bin #Dihedral Angles> [Cluster 1]
. .
. .
. .
<Cluster #>  <Weight>
<Bin1><Bin2>...<Bin #Dihedral Angles> [# Clusters]
```

The first line is the number of dihedral angles that will be binned, following the definition of those dihedral angles (4 atoms using sander atom #s, starting from 1) and the bin size for each dihedral angle. Next is the total # of clusters followed by lines providing information about each cluster: the cluster number, weight and ID as defined by dihedral binning. The ID is composed of consecutive 3 digit integers, 1 for each dihedral angle. For example, a structure belonging to cluster 7 with a weight of 2 with 2 dihedral angles that fall in bins 3 and 8 would look like:

```
7 2 003008
```

21.5.4. Hamiltonian replica exchange

Instead of spacing replicas throughout temperature space, you can also space replicas throughout "Hamiltonian space." That is, every replica has a different Hamiltonian, or energy function, and exchange attempts occur between adjacent Hamiltonians. With *sander* and *pmemd*, Hamiltonian replica exchange is implemented by exchanging coordinates between replicas and evaluating the energy of that new structure. The corresponding detailed balance equation that is used to compute the exchange probability is shown in Eq. 21.11. This option is enabled by using *-rem 3* on the command-lines in the groupfile.

$$P_{i\to j} = \min\{1, \exp\left(-\beta_1\left[H_1(x_2) - H_1(x_1)\right] - \beta_2\left[H_2(x_1) - H_2(x_2)\right]\right) \tag{21.11}$$

Here, state i refers to the replica combination $[\beta_1 H_1(x_1), \beta_2 H_2(x_2)]$ and state j refers to the replica combination $[\beta_1 H_1(x_2), \beta_2 H_2(x_1)]$. Eq. 21.11 assumes that only coordinates are traded between exchanging replicas, but allows for the temperatures to differ. The temperature does not exchange upon a successful attempt, but velocities are swapped following successful exchange attempts and scaled by $\sqrt{T_{new}/T_{old}}$ to match the target temperature of their new replica.

21.5.4.1. Free Energy Perturbation

Upon closer inspection of Eq. 21.11, we can see a close resemblance to Free Energy Perturbation[433, 438]

$$\Delta G_{a\to b} = -k_B T \ln\left[\exp\left(-\beta\left(E_b - E_a\right)\right)\right] \tag{21.12}$$

We can see that for every exchange attempt, the required ΔE is calculated in both directions. The value for the free energy (in both directions) is accumulated and reported in the rem.log file each time an exchange is attempted.

For replica exchange free energy perturbation (REFEP), multiple topology files are often needed that correspond to a value of an alchemical parameter, λ, similar to thermodynamic integration. The ParmEd program included with AmberTools can be used to generate the intermediate topology files by scaling charges and/or van der Waals parameters. In this case, because coordinates are exchanged, each replica tracks a particular Hamiltonian and set of control variables, rather than a sequence of configuations. Note this is the opposite behavior of T-REMD in which replicas change temperatures but keep the same sequence of configurations.

21.5.4.2. Umbrella Sampling

Hamiltonian exchange can be used to perform replica exchange umbrella sampling [439] using the NMR flat well restraints. In this case, every line of the group file needs a different restraint file in which the center of the biasing umbrella changes. Each replica tracks a particular umbrella location rather than a replica trajectory. Note this is the opposite behavior of T-REMD in which replicas change temperatures but keeps the same replica trajectory.

21.5.4.3. Steps for running H-REMD simulations

Note: before running Hamiltonian replica exchange (H-REMD), you should be familiar with Temperature replica exchange (T-REMD) simulations. H-REMD simulations are set up similarly to T-REMD simulations. Each replica is specified on a line of a groupfile and is run with *multisander*. Each replica differs either by simulation control parameters in the input file (*e.g.*, for umbrella sampling replica exchange[439] or REXAMD [440, 441]) or parameters in the topology file (*e.g.*, REFEP).

- The majority of H-REMD settings are similar to T-REMD. A groupfile is needed. The number of replicas must be an even number (so that all replicas have a partner for exchange). Constant pressure is not supported for REMD simulations. This means `ntp` must be 0.
- Depending on the type of H-REMD, all replicas may have different force fields/control variables (if the differences are too large, the exchange probability may suffer)
- The order of the replicas in the groupfile is very important. As a general rule in all H-REMD simulations, the least different Hamiltonians (replicas) should be neighbors. Because this method is relatively new, there are very limited discussions in the literature about the optimum positions of replicas in the Hamiltonian ladder [442, 443]. Exchange neighbors are defined by adjacent lines in the groupfile (i.e., each replica exchanges 'right' or 'up' with the replica defined by the line above and exchanges 'left' or 'down' with the replica defined by the line below in the groupfile).
- For editing the prmtop, e.g., in the case of REFEP, there is a python script in AmberTools, *parmed*, which facilitates the modifications of Amber topology files. See *Section 14.2* for details.
- In H-REMD, each replica has a different Hamiltonian. In contrast to T-REMD, neighbor replicas exchange their conformations, which means each replica keeps its initial Hamiltonian and there is no need for post-processing (i.e., using *ptraj* or *cpptraj*) to extract sub-ensembles. However, you will have to post-process in order to reconstruct replica-based time series.

To enable H-REMD the *-rem* flag on the command-line must be given the value *3*. H-REMD simulations require the same input files as T-REMD simulations and generates the same output files. The output printed in the remlog file differs significantly from that found in the remlog file for T-REMD, however. Example remlog output for H-REMD is shown below:

```
# Replica Exchange log file
# numexchg is 10000
# REMD filenames:
#   remlog= remlog
#   remtype= rem.type
# Rep#, Neibr#, Temp0, PotE(x_1), PotE(x_2), left_fe, right_fe, Success, Success rate (i,i+1)
# exchange        1
     1     8    300.00 -12783.23 -12755.40    -16.39      0.00    F        0.00
     2     3    300.00 -12839.84 -12802.56      0.00     -0.05    T        2.00
     3     2    300.00 -12802.60 -12839.79     -0.04      0.00    T        0.00
     4     5    300.00 -12847.41 -12858.37      0.00     -0.78    F        0.00
     5     4    300.00 -12858.19 -12846.63      0.18      0.00    F        0.00
     6     7    300.00 -12859.65 -12833.42      0.00      0.30    T        2.00
     7     6    300.00 -12833.63 -12859.95     -0.21      0.00    T        0.00
     8     1    300.00 -12771.63 -12766.84      0.00    -16.23    F        0.00
# exchange        2
     1     2    300.00 -12825.03 -13147.73      0.00     -0.62    F        0.00
     2     1    300.00 -13148.20 -12824.42     -0.47      0.00    F        1.00
     3     4    300.00 -13136.97 -12823.77      0.00      0.62    T        1.00
     4     3    300.00 -12823.32 -13137.59      0.44      0.00    T        0.00
     5     6    300.00 -12919.25 -13181.18      0.00     -0.41    T        1.00
     6     5    300.00 -13180.48 -12918.84      0.70      0.00    T        1.00
     7     8    300.00 -13162.39 -12775.37      0.00      0.16    T        1.00
     8     7    300.00 -12775.59 -13162.55     -0.22      0.00    T        0.00
...
```

The columns, in order, are the replica number, the exchange partner for this attempt, the target temperature, the potential energy of the current structure, the potential energy of the proposed structure, the free energy difference calculated via Eq. 21.12 for all exchanges to the 'left' (or 'up' in the Hamiltonian ladder), all free energies for exchanges to the 'right' (or 'down' in the Hamiltonian ladder), whether the exchange attempt succeeded (T) or not (F), and the average success rate. For each step, the only free energy values printed are those between replicas that attempted to exchange. All free energies between non-exchanging pairs are set to 0 for that step. Therefore, the 'final' free energies can be found by summing the respective terms from the last two exchanges in the *remlog* file. All energies have units of kcal/mol, and temperatures have units of Kelvin.

21.5.4.4. An example

When running H-REMD, the format of the groupfile is very similar to that in T-REMD, but specific details depend on the type of simulation being performed. In the case of REFEP, the *groupfile* may look like the following:

```
-O -i mdin -p prmtop.0 -c inpcrd.0 -suffix 000
-O -i mdin -p prmtop.1 -c inpcrd.1 -suffix 001
-O -i mdin -p prmtop.2 -c inpcrd.2 -suffix 002
-O -i mdin -p prmtop.3 -c inpcrd.3 -suffix 003
-O -i mdin -p prmtop.4 -c inpcrd.4 -suffix 004
-O -i mdin -p prmtop.5 -c inpcrd.5 -suffix 005
-O -i mdin -p prmtop.6 -c inpcrd.6 -suffix 006
-O -i mdin -p prmtop.7 -c inpcrd.7 -suffix 007
```

Notice how the topology file differs in each case, but the input file remains the same. An example *groupfile* for umbrella sampling may look like the following:

```
-O -i mdin.0 -p prmtop -c inpcrd.0 -suffix 000
-O -i mdin.1 -p prmtop -c inpcrd.1 -suffix 001
-O -i mdin.2 -p prmtop -c inpcrd.2 -suffix 002
-O -i mdin.3 -p prmtop -c inpcrd.3 -suffix 003
-O -i mdin.4 -p prmtop -c inpcrd.4 -suffix 004
-O -i mdin.5 -p prmtop -c inpcrd.5 -suffix 005
-O -i mdin.6 -p prmtop -c inpcrd.6 -suffix 006
-O -i mdin.7 -p prmtop -c inpcrd.7 -suffix 007
```

Notice in this case how the topology file is the same but the input file differs in each case (which is where the center of the umbrella is defined). Like T-REMD, *sander.MPI* (or *pmemd.MPI*) are executed via the following command:

```
mpirun -np 16 sander.MPI -ng 8 -groupfile groupfile -rem 3
```

Note that the particular method for launching an MPI program may depend on your MPI implementation. Also, *pmemd* requires at least 2 threads per replica, whereas *sander* will work with just 1.

21.5.5. RXSGLD: Replica exchange using Self-Guided Langevin Dynamics

RXSGLD utilizes the guiding force, *sgft* or *tempsg*, to define replicas. SGLD simulations are performed for replicas[388]. Please refer to Section 20.1 about how to set up SGLD simulations. When temperature is the same for all replicas, the replica exchange ratios are high and so is the conformational search efficiency. RXSGLD is an alternative to SGLD, SGLDfp, or SGLDg to achieve efficient conformational search while being able to obtain the canonical ensemble distribution. RXSGLD is turned on if *isgld* is set to 1 or 3 in the *sander* or *pmemd* input file when performing replica exchange simulations (i.e., *rem* > 0).

For the convenience of reference, we define replicas as non-interacting identical simulation systems and define stages as simulation conditions between which replicas transit. In T-REMD, stages are different by temperatures, while in RXSGLD, stages are different by the strength of guiding forces, as defined by *sgld* or *tempsg*. For example, we can set *tempsg*=300, 310, 325, 345, 370, 400, 440, and 500K for stages 1 to 8, respectively, while *temp0*=300K for all stages. In RXSGLD, temperatures in different stages can be the same or different from each other; however, it is prefered to keep all temperatures the same to achieve high replica exchange efficiency.

Because the temperatures of different stages may be the same, the stages are given a ID from 1 to Nrep. Like T-REMD, each RXSGLD trajectory file is for each replica. Unlike T-REMD, the frames of RXSGLD trajectories are proceeded by the following information:

```
RXSGLD <replica#> <exchange#> <step#> <stage ID>
```

The output from RXSGLD contains the following lines:

```
TEMP0= <temp0> SGFT= <sgft> TEMPSG= <tempsg> STAGE= <stag ID> REPNUM= <rep#> EXCHANGE= <exchange#>
```

RXSGLD trajectories can be processed with PTRAJ analogously to T-REMD trajectories: merely replace temperatures with stage IDs. For example, to extract the trajectory on stage 1, we can use the following command:

```
ptraj rxsgld.top <<EOF
trajin rxsgld.trj.000 rxsgldtraj rxsgldid 1
trajout rxsgld.trj.stag 1
EOF
```

21.5.6. pH-REMD

In constant pH REMD, replicas attempt to exchange their solution pH values in much the same way as temperatures are exchanged in T-REMD. The idea of swapping pH in the discrete protonation state implementation (described in Section 22) was proposed by [Itoh et al. 434] and later implemented and evaluated in Amber. [435] The implementation here works very similarly to T-REMD, except each replica is given a different value of `solvph` in the mdin file instead of `temp0`. The exchange probability, shown in Eq. 21.13, is derived under the assumption that all replicas have the same temperature.

$$P_{i\to j} = \min\left\{1, \exp\left[\ln 10\left(N_i^{H^+} - N_j^{H^+}\right)(pH_i - pH_j)\right]\right\} \tag{21.13}$$

Where $N_i^{H^+}$ is the number of titratable protons currently ‘active’ in state i.

Before running pH-REMD simulations, you should first be familiar with running constant pH MD described in Section 22, since it will help you set up each replica. Aside from the changes required to run at constant pH, setting up pH-REMD simulations is quite similar to setting up T-REMD simulations. Each replica should have the same topology file, and all mdin files should be identical except for the value of `solvph`. Furthermore, each residue should be titrating the same residues (this is very important). For instance, you cannot turn ‘off’ carboxylate titrations at basic pH if your pH-REMD spans both acidic and basic conditions.

21.5.6.1. Analyzing Output

The output from pH-REMD simulations is analyzed in the same way as standard constant pH simulations, with some preprocessing required. Because the pH of each replica changes upon successful replica exchange attempts, each replica contains members from ensembles at all pHs. Therefore, you must use ptraj or cpptraj to extract ensembles at each pH. To simplify the coding required, the pH is stored as the ‘temperature’ in each trajectory, so the T-REMD machinery should be used in *ptraj*/*cpptraj* to extract desired ensembles.

The cpout files have additional information added to them to indicate which protonation states belong to which ensembles. This is done by printing the pH next to each record, as shown below.

```
Solvent pH:  3.00000
Monte Carlo step size:        5
Time step:        5
Time:     10.008
Residue    0 State:  3 pH:   3.000

Residue    0 State:  3 pH:   3.000

Residue    0 State:  3 pH:   3.500

Residue    0 State:  3 pH:   3.500

Residue    0 State:  3 pH:   2.000

Residue    0 State:  3 pH:   2.000

Residue    0 State:  3 pH:   2.500
```

You can see that the pH is changing between snapshots. In addition to generating pH-based trajectories, you also must generate pH-based `cpout` information that is stored in each `cpout` file. The replica pH is identified on each line of the cpout file to aid in constructing the pH-specific protonation state ensembles. To aid with this, the `cphstats` program, described in Subsection 22.7.5, has an option to provide a “prefix” that defines the new file names for the pH-specific `cpout` files that it builds from a list of *each cpout file from a* ***single*** *pH-REMD simulation*. For example:

```
cphstats --fix-remd 1AKI.cpout 1AKI.cpout.000 1AKI.cpout.001 \
                               1AKI.cpout.002 1AKI.cpout.003
```

Assuming you ran replicas at pH 2, 3, 4, and 5, this will generate files 1AKI.cpout.pH_2.00, 1AKI.cpout.pH_3.00, 1AKI.cpout.pH_4.00, and 1AKI.cpout.pH_5.00, with their respective ensembles. If you ran, for instance, 20 ns of simulation in chunks of 5 ns (so you ran 4 “chunks” after 3 restarts), you will need to run this command 4 times—once for each simulation segment. You should analyze the resulting protonation state distribution using these newly-generated `cpout` files.

21.5.6.2. A pH-REMD Example

Below is an example in which 4 replicas are run at pH values of 2.0, 2.5, 3.0, and 3.5. The command below shows an example of running this simulation on 8 processors (2 processors for each replica). Note that running MPI programs may vary from computer to computer.

```
mpirun -np 8 sander.MPI -ng 4 -groupfile groupfile -rem 4 -remlog rem.log
```

The groupfile in this example is shown below:

```
-O -i phremd.pH2.0.mdin -cpin cpin -p ASPREF.top -c ASPREF.rst7
-O -i phremd.pH2.5.mdin -cpin cpin -p ASPREF.top -c ASPREF.rst7
-O -i phremd.pH3.0.mdin -cpin cpin -p ASPREF.top -c ASPREF.rst7
-O -i phremd.pH3.5.mdin -cpin cpin -p ASPREF.top -c ASPREF.rst7
```

The suffixes “.000”, “.001”, “.002”, and “.003” will be added to the output files from each of the replicas in order to distinguish them from each other by default. This suffix can be changed using the “-suffix” flag for that replica. Any suffix provided this way will be applied to ALL output files, regardless of whether or not they are specified. This is only true for multi-sander and multi-pmemd simulations.

The resulting `rem.log` file looks like the following:

```
# Replica Exchange log file
# numexchg is         50
# REMD filenames:
#   remlog= rem.log
#   remtype= rem.type
# Rep#, N_prot, old_pH, new_pH, Success rate (i,i+1)
# exchange        1
     1       1   2.000  3.500   0.0000
     2       1   2.500  3.000   2.0000
     3       1   3.000  2.500   0.0000
     4       1   3.500  2.000   2.0000
# exchange        2
     1       1   3.500  3.000   1.0000
     2       1   3.000  3.500   1.0000
     3       1   2.500  2.000   1.0000
     4       1   2.000  2.500   1.0000
# exchange        3
     1       1   3.000  2.500   0.6667
     2       1   3.500  2.000   1.3333
     3       1   2.000  3.500   0.6667
     4       1   2.500  3.000   1.3333
# exchange        4
     1       1   2.500  2.000   1.0000
     2       1   2.000  2.500   1.0000
     3       1   3.500  3.000   1.0000
     4       1   3.000  3.500   1.0000
```

The columns are the current replica number (which never changes because pH is swapped between replicas), the total number of protons “active” on all of the titratable sites, the original solution pH, the new solution pH after exchange, and the success ratio (multiplied by 2 to account for swapping neighbors each exchange attempt).

21.5.7. Multi-dimensional Replica Exchange

Multi-dimensional replica exchange refers to an expanded ensemble technique in which each subensemble (i.e., each replica) is defined by multiple state parameters such as the temperature of the heat bath or the Hamiltonian. For such systems, an exchange attempt between two arbitrary replicas yields a complex equation for the exchange probability that must be derived and specified for each type of multi-dimensional exchange scheme. However, if exchange attempts between replica pairs are restricted to pairs that only differ in *one* state parameter, such as the temperature, then the exchange probability equation reduces to the one used for that particular type of replica exchange. Such an approach allows the existing exchange routines to be used in complex, multi-dimensional replica exchange simulations.

To implement the scheme described above, the entire expanded ensemble is subdivided into *dimensions* which are further divided into *groups*. Each dimension is defined by a particular type of exchange attempt—namely temperature exchange, Hamiltonian exchange, or pH exchange—and assigns every replica in the simulation to a particular *group*. Each group is like a one-dimensional REMD simulation by itself—replicas differ only by the one thermodynamic state parameter used to define that particular dimension. Exchange attempts occur only between nearest neighbors of a single replica group.

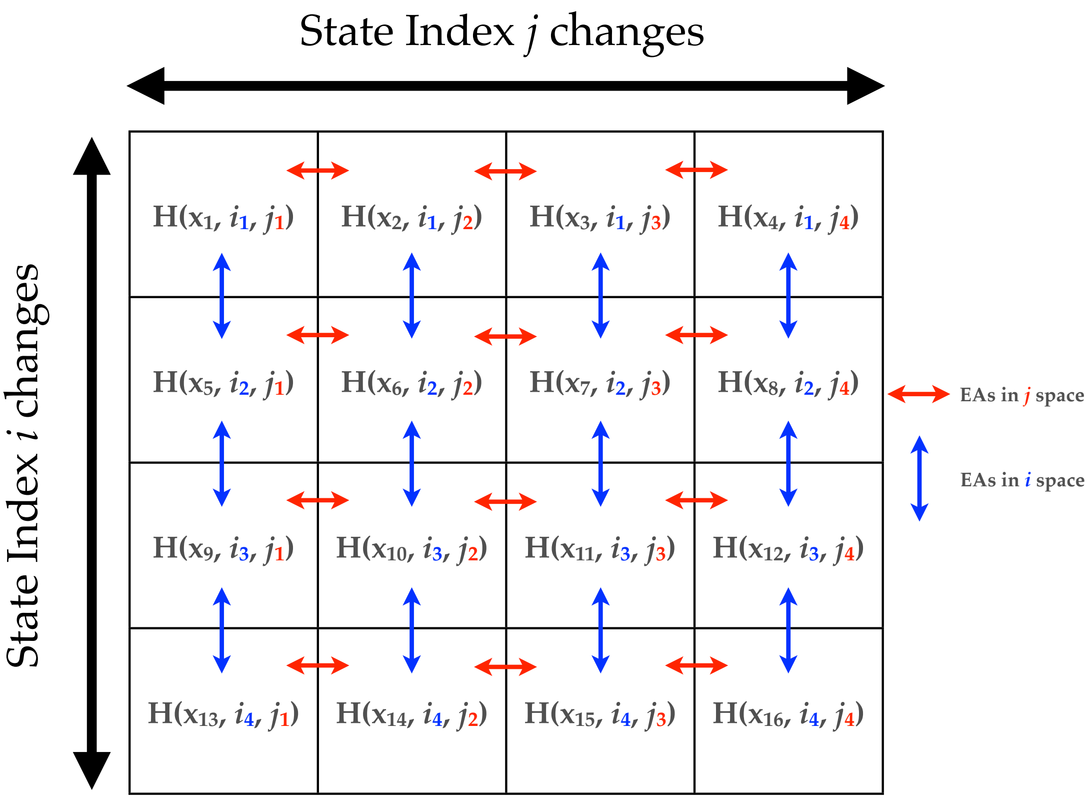


Figure 21.2.: *Schematic showing exchange attempts (EAs) in multi-dimensional REMD simulations. Exchange attempts are indicated by the colored arrows, where red arrows indicate exchange attempts between replicas in a group of the dimension defined by the* j *state parameters. Blue arrows indicate exchange attempts between replicas in the dimension defined by the state parameter* i*. Figure taken from ref. 444.*

No two replicas should be in the same group in more than one dimension since, by definition, that would require the state parameters of those two replicas to differ in more than one dimension (which would preclude them from being part of the same group in *any* dimension according to the scheme described previously). It is easiest to understand this scheme in the context of a 2-dimensional replica exchange ensemble with replicas represented by elements of a matrix, as in Fig. 21.2.

21.5.7.1. Running multi-dimensional replica exchange simulations

Multi-dimensional REMD simulations require an extra input file provided on the command-line (*not* in the groupfile) that defines each replica group in each dimension. Every replica must be assigned to one and only one group in every dimension (failure to do so results in an error message). Furthermore, each group must consist of an even number of replicas. Dimensions are defined in the `&multirem` namelist in the REMD input file. Each `&multirem` namelist adds another dimension. The following variables may be specified in the `&multirem` namelist:

group(:, :) 2-dimensional (Fortran-style) array defining the group (first dimension) and the position within that group (second dimension). See the description of the exchange routines above to see if the ordering within each group is important (for example, the ordering defines exchange partners in H-REMD while replicas are automatically sorted by target temperature in T-REMD). Indexes in this array start from 1, and index *n* corresponds to the n^{th} replica defined in the groupfile. The suggested syntax for assigning to this variable is shown in the example below.

exch_type Defines the type of exchange that will be performed. Supported values (case-insensitive) are “temperature” (or “temp”), “Hamiltonian” (or “HREMD”), and “pH.” pH-REMD is not currently compatible with T-REMD in another dimension. Also, H-REMD/pH-REMD simulations are not expected to work well. pH-compatible temperature and umbrella exchange are planned for the future.

desc Description that will be printed in the rem.log files. This is for documentation purposes only, and will have no effect on the simulation.

A sample input file that performs alchemical Hamiltonian-REMD in one dimension and Temperature-REMD in another dimension is shown below.

```
Temperature REMD
&multirem
   exch_type = 'TEMPERATURE',
   group(1,:) = 1,2,
   group(2,:) = 3,4,
   desc = 'Temperature exchange from 300K to 400K'
/
Hamiltonian REMD
&multirem
   exch_type='HAMILTONIAN',
   group(1,:) = 1,3,
   group(2,:) = 2,4,
   desc = 'Protonated ASP to Deprotonated ASP mutation'
/
```

Running multi-dimensional REMD simulations differs from running them in a single dimension. First, restarts and trajectories *must* be written in the NetCDF format (ntxo=2 for restarts and ioutfm=1 for trajectories). These changes are applied by default for multi-dimensional REMD simulations. Next, the REMD input file is taken following the `-remd-file` flag, and `-rem` should not be specified (it is set to -1 internally when `-remd-file` is read). An example command-line corresponding to the 4-replica example input file is shown below:

```
mpirun -np 4 sander.MPI -ng 4 -groupfile groupfile \
                        -remd-file remd.dim -remlog rem.log
```

The replica exchange information is stored in the remlog files written during the simulation. A separate remlog file is written for each dimension with the name <prefix>.*n* where *n* is the n^{th} dimension read from the REMD input file and <prefix> is the file name given on the command-line for the `-remlog` switch.

21.5.7.2. Restarting multi-dimensional replica exchange simulations

Some exchange types swap state parameters (e.g., temperature and pH) while others swap coordinates (Hamiltonian), meaning that the ordering of the groupfile may change for restarts (see Fig. 21.3). To prevent requiring you to rewrite a new REMD file or groupfile each restart, the group number and replica position for each dimension is stored in the restart file. When irest=1 (see page 317) and the restart files contain the replica position information, the position of each replica in each dimension is set to the values stored in the restart file. This allows the same groupfile and REMD file to be used for every subsequent restart. For general information on restarting a REMD simulation seeSubsection 21.5.3.1.

Note, if the REMD index information is not present in the restart file *or* the REMD dimension information in the restart does not match what is defined in the REMD input file, the replica ordering will be assigned as it is defined in the REMD input file.

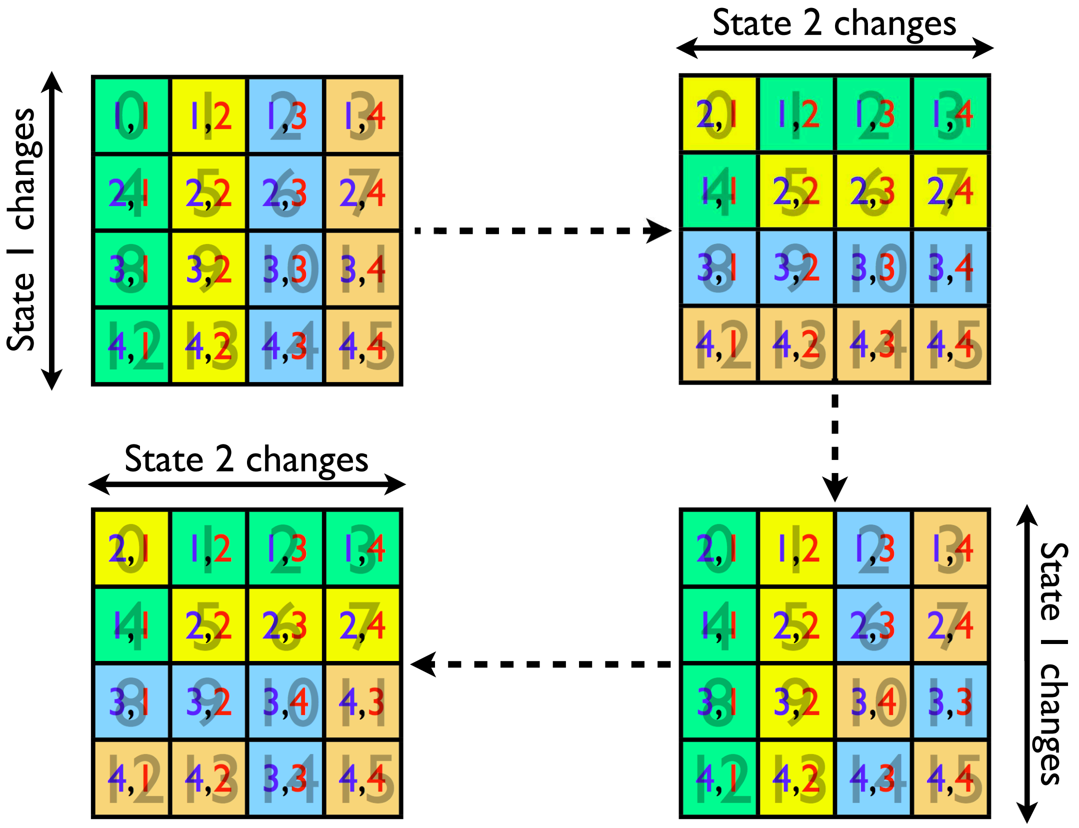

State 1 changes
State 2 changes
State 2 changes
State 1 changes

Figure 21.3.: *Replica arrangement in multi-dimensional REMD simulations at multiple exchange steps following some successful state parameter exchanges. A large gray number in the background is the original placement in the REMD input file. A blue and red number pair is the group number and position in the group, respectively. Replicas with the same color are part of the same group. Figure taken from ref. 444.*

21.5.7.3. Analyzing multi-dimensional replica exchange simulations

The REMD log file for each dimension is further divided into the log messages for each group, as shown below.

```
# Replica Exchange log file
# numexchg is        100
# Dimension    1 of    2
# Description: Temperature exchange from 300K to 400K
# exchange_type = TEMPERATURE
# REMD filenames:
# remlog= rem.log.1
# remd dimension file= remd.dim
# Rep#, Velocity Scaling, T, Eptot, Temp0, NewTemp0, Success rate (i,i+1), ResStruct#
# exchange        1 REMD group        1
 1     -1.00      0.00    -40.69    300.00    300.00      0.00       0
 2     -1.00      0.00    -19.77    400.00    400.00      0.00       0
# exchange        1 REMD group        2
 1     -1.00      0.00    -66.78    300.00    300.00      0.00       0
 2     -1.00      0.00    -46.06    400.00    400.00      0.00       0
# exchange        3 REMD group        1
 1     -1.00    221.66    -34.00    300.00    300.00      0.00       0
 2     -1.00    347.52    -29.09    400.00    400.00      0.00       0
# exchange        3 REMD group        2
 1     -1.00    257.14    -63.10    300.00    300.00      0.00       0
 2     -1.00    332.76    -26.73    400.00    400.00      0.00       0
```

The above example is shown for the first dimension (temperature) of the example REMD file shown in Sec. 21.5.7.1. The columns are the same as those used in the corresponding 1-dimensional REMD simulation for that exchange type.

To analyze structural properties, you must use *cpptraj* to properly snapshots into the appropriate replicas. See Chapter 28 for more details.

21.6. Adaptively Biased MD, Steered MD, Umbrella Sampling with REMD and String Method

21.6.1. Overview

The following describes a suite of modules useful for the calculation of the free energy associated with a reaction coordinate $\sigma(\mathbf{r}_1,\ldots,\mathbf{r}_N)$ (which is defined as a smooth function of the atomic positions $\mathbf{r}_1,\ldots,\mathbf{r}_N$):

$$f(\xi) = -k_B T \ln \left\langle \delta\left[\xi - \sigma(\mathbf{r}_1,\ldots,\mathbf{r}_N)\right] \right\rangle,$$

(the angular brackets denote an ensemble average, k_B is the Boltzmann constant and T is the temperature) that is also frequently referred to as the *potential of mean force*.

Specifically, new frameworks are provided for equilibrium umbrella sampling and steered molecular dynamics that enhance the functionality delivered by earlier implementations (described earlier in this manual), along with a new Adaptively Biased Molecular Dynamics (ABMD) method [445] that belongs to the general category of umbrella sampling methods with a time-dependent potential. Such methods were first introduced by Huber, Torda and van Gunsteren (the Local Elevation Method [446]) in the molecular dynamics (MD) context, and by Wang and Landau in the context of Monte Carlo simulations [447]. More recent approaches include the metadynamics method [448, 449]. All these methods estimate the free energy of a reaction coordinate from an evolving ensemble of realizations, and use that estimate to bias the system dynamics to flatten an effective free energy surface. Collectively, these methods may all be considered to be umbrella sampling methods with an evolving potential. The algorithms discussed here were developed by the group of Prof. Celeste Sagui (sagui@ncsu.edu) and Prof. Christopher Roland (cmroland@ncsu.edu); the current version was implemented by Dr. Volodymyr Babin.

The `ABMD` method grew out of attempts to speed up and streamline the metadynamics method for free energy calculations with a *controllable* accuracy. It is characterized by a favorable scaling in time, and only a few (two) control parameters. It is formulated in terms of the following equations:

$$m_a \frac{d^2\mathbf{r}_a}{dt^2} = \mathbf{F}_a + \frac{\partial}{\partial \mathbf{r}_a} U\left[t|\sigma(\mathbf{r}_1,\ldots,\mathbf{r}_N)\right],$$

$$\frac{\partial U(t|\xi)}{\partial t} = \frac{k_B T}{\tau_F} G\left[\xi - \sigma(\mathbf{r}_1,\ldots,\mathbf{r}_N)\right],$$

where the first equation represents Newton's law that governs ordinary `MD` (temperature and pressure regulation terms are not shown) augmented with an additional force coming from the time dependent biasing potential $U(t|\xi)$ [$U(t=0|\xi)=0$], whose time evolution is given by the second equation. $G(\xi)$ is a positive definite and symmetric kernel, which may be thought of as a smoothed Dirac delta function. For large enough τ_F (the flooding timescale) and small kernel width, the biasing potential $U(t|\xi)$ converges towards $-f(\xi)$ as $t \to \infty$.

Our numerical implementation of the `ABMD` method involves the use of a bi-weight kernel along with the use of cubic B-splines (or products thereof) to discretize the biasing potential $U(t|\xi)$ w.r.t. ξ, and an Euler-like scheme for time integration. `ABMD` admits two important extensions, which lead to a more uniform flattening of $U(t|\xi)+f(\xi)$ due to an improved sampling of the "evolving" canonical distribution. The first extension is identical in spirit to the *multiple walkers metadynamics* [450, 451]. It amounts to carrying out several different `MD` simulations biased by the same $U(t|\xi)$, which evolves via:

$$\frac{\partial U(t|\xi)}{\partial t} = \frac{k_B T}{\tau_F} \sum_{\alpha} G\left[\xi - \sigma(\mathbf{r}_1^{\alpha},\ldots,\mathbf{r}_N^{\alpha})\right],$$

where α labels different `MD` trajectories. A second extension is to gather several different `MD` trajectories, each bearing its own biasing potential and, if desired, its own distinct collective variable, into a generalized ensemble for "replica exchange" with modified "exchange" rules [452–454]. Both extensions are advantageous and lead to a more uniform flattening of $U(t|\xi)+f(\xi)$.

In order to assess and improve the accuracy of the free energies, the `ABMD` accumulations may need to be followed up with equilibrium umbrella sampling runs, which make use of the biasing potential $U(t|\xi)$ as is. Such a procedure is very much in the spirit of adaptive umbrella sampling. With these runs, one calculates the biased probability density:

$$p^B(\xi) = \left\langle \delta\left[\xi - \sigma(\mathbf{r}_1,\ldots,\mathbf{r}_N)\right]\right\rangle_B.$$

The idea here is that if, as a result of an `ABMD` run, $f(\xi)+U(t|\xi)=0$ exactly, then the biased probability density $p^B(\xi)$ would be flat (constant). In practice, this is typically not the case, but one can use $p^B(\xi)$ to "correct" the free energy via:

$$f(\xi) = -U(\xi) - k_B T \ln p^B(\xi).$$

With the ABMD procedure, one can obtain accurate free energy curves and equilibrium properties. We note that to obtain ABMD free energies requires a (minor) amount of post-processing by means of the nfe-umbrella-slice utility freely available in AmberTools as described in Subsection 21.6.8. This methodology has been applied to a variety of biomolecular systems, including small peptides [445, 455, 456], sugar puckering [457], polyproline systems [458–460], guest-host systems [461, 462], polyglutamine systems [463, 464], and DNA systems [465]. In addition, SMD simulations (discussed below) have been used to examine transition pathways and mechanisms, to estimate free energy differences [459, 466], and to calculate transition rates [467–469].

While the above represents the basic ABMD implementation, AMBER16 introduces two additional algorithms – the Well-Tempered (WT) ABMD and a selection mechanism for multiple walker ABMD – both of which enhance the stability and convergence of an ABMD simulation. Future enhancements to the set ABMD simulation tools are to include the Driven ABMD (D-ABMD) [470] and the Swarms-of-Trajectories String Method (STSM) [471]. Current version of these codes were inplemented by Dr. Mahmoud Moradi (moradi@uark.edu) and Feng Pan (fpan3@ncsu.edu).

The Well-Tempered ABMD: An alternative to the follow-up equilibrium simulations for increased ABMD accuracy is provided by the WT-ABMD, which is implemented in the spirit of the WT-metadynamics [472]. In the original ABMD implementation, the history dependent biasing potential is built up at a fixed rate:

$$U(\xi,t) = U^0(\xi) + \int_0^t dt'\, \omega\, G(\xi - \xi^{t'}), \tag{21.14}$$

in which $U(\xi,t)$ is the biasing potential at time t, U^0 is an arbitrary function that typically represents the initial guess for the biasing potential (in the absence of a guess, this is assumed to be flat) and $\omega = k_B T/\tau_F$ is a constant, unbiased rate. As the simulation proceeds and reaches convergence, then $\langle U(\xi, t \to \infty)\rangle_a \approx U^s(\xi) + u(t)$, in which $\langle . \rangle_a$ is the ensemble-average over the adaptive trajectories, the stationary term is $U^s \approx -F(\xi)$, and $u(t)$ is an additive time-dependent constant [472]. Unfortunately, updating the biasing potential at the same rate throughout the simulation may lead to a poorly converged result, since the biasing potential ends up fluctuating around $-F(\xi)$ with an amplitude that depends on ω.

One way to resolve this problem is to update the kernel at a non-uniform rate by means of a "well-tempered" ω:

$$U(\xi,t) = U^0(\xi) + \int_0^t dt'\, \omega(\xi^{t'},t')\, G(\xi - \xi^{t'}), \tag{21.15}$$

in which $\omega(\xi,t)$ is a time-dependent, non-uniform rate chosen to be $\omega_0\, \mathrm{e}^{-\beta' U(\xi,t)}$ ($1/\beta' = k_B T'$ where T' is a pseudo-temperature) that reduces to a constant ω_0 in the $\beta' \to 0$ limit (*i.e.*, resulting in conventional ABMD). With this choice, one can show that $\langle U(\xi, t \to \infty)\rangle_a \approx U^s(\xi) + u(t)$, ($u(t)$ is an additive constant) in which $U^s(\xi)$ and $F(\xi)$ are related via $U^s(\xi) = -(1+\frac{\beta'}{\beta})^{-1} F(\xi)$ or $F(\xi) = -(1+\frac{T}{T'}) U^s(\xi)$. This way of updating the biasing potential leads to a considerably smoother convergence to the desired free energy and more stable ABMD simulations.

Multiple walker selection algorithm: The ABMD multiple walker algorithm can be improved by allowing for periodic interactions between the different walkers and "resampling" on-the-fly. The rationale behind this is that not all walkers are equally effective in sampling the configuration space. A situation that is all too common is that different walkers end up being “bunched up" or clustered together in some local metastable region, because of hidden barriers that are oriented along orthogonal degrees of freedom to the reaction coordinate. To improve this situation, one would like to facilitate walkers that are sampling the undersampled regions of phase space, and force the walkers in the oversampled regions to move away and explore regions not yet covered. Such an algorithm has previously been implemented via scripts in the NAMD code for the adaptive biasing force algorithm [473].

A resampling or selection algorithms for interacting multiple walkers requires a continual monitoring of the walkers by means of a periodic evaluation of a fitness function and a resampling of the walkers according to their fitness efficiency[473]. Efficient walkers that are wandering in the undersampled regions are enhanced by being cloned, while inefficient walkers found in the oversampled regions of phase space are correspondingly killed. This procedure is then repeated periodically during the simulation, thereby accelerating convergence to a more uniform distribution of walkers and flattening of the free energy landscape.

Our specific interacting/resampling/selection multiple-walker algorithm is implemented as follows. Each walker n is assigned a weight w_n, which is evaluated at the end of each resampling period of time τ. At the i^{th} resampling period, i.e., from time $t_{(i-1)} = (i-1)\tau$ to $t_i = i\tau$, walker n moves through configuration space building up its own trajectory $(r_1^n, \ldots, r_N^n)$. The weights are then tested and updated every fixed time interval of length τ. Specifically, after the i^{th} time interval, weights are estimated by:

$$w_n = K^{-1} \exp\left(\int_{t_{i-1}}^{t_i} S(\xi_n^t) dt \right),$$

where ξ_n^t represents the collective variable evaluated at time t for trajectory n, $K = \sum_{n=1}^{N_w} w_n$ is the normalization factor, and

$$S(\xi) = C\nabla^2(\rho(\xi))/\rho(\xi),$$

with $\rho(\xi)$ representing the density of microstates in the collective variable space and C a constant. The quantity $S(\xi)$ will be positive typically if the walker is found in the undersampled regions, which have a convex density function. Similarly, a negative $S(\xi)$ value indicates that the system is in the concave region of the density function, which typically is oversampled. In the context of ABMD implementation, the biasing potential is approximately proportional to the histogram of the collective variable by construction, and represents a good estimate for ρ. The implementation is therefore straightforward; the integral above is estimated for each trajectory independently by summing over $S(\xi_n^t)$ at every step from $t = t_{i-1}$ to $t = t_i$, in which Δt is the MD *timestep*. At the end of each period the walkers send their unnormalized weight estimates to the "master processor" to normalize them. A stochastic resampling method is then used to clone/kill the replicas based on their weight factors [473]. The number of copies present in the next period for walker n is determined by the integer number:

$$\begin{aligned} W_1 &= \lfloor \eta_1 + N_w w_1 \rfloor, \\ W_n &= \lfloor \eta_n + N_w \sum_{m=1}^{n} w_m \rfloor - \lfloor \eta_n + N_w \sum_{m=1}^{n-1} w_m \rfloor, \quad \text{for } n > 1. \end{aligned}$$

in which $0 < \eta_n < 1$ is drawn from a uniform distribution (using a random number generator). The atomic coordinates and velocities of the walkers with $N_n > 0$ are "sent" to N_n walkers. The resampling algorithm above guarantees $\sum_n W_n = N_w$.

In terms of an ABMD simulation, the selection algorithm is most beneficial during the initial and middle parts of the simulation when there are large variations in the biasing potential. In the latter parts, when the effective free energy is almost flat, the distribution of walkers should be roughly uniform. In that case, the selection mechanism is unnecessary and, if one wishes to continue the simulation, it is best to proceed with the non interacting multiple walker algorithm. It has been found that a convenient stopping mechanism may be based on the entropy of the weights. Defining $H = \sum_n w_n log(w_n)$, the selection mechanism will be stopped if $E_w = H - log(1/N_w)$ goes below -$\varepsilon log(1/N_w)$. Here, $log(1/N_w)$ represents the entropy of uniform weights, and the stopping parameter ε varies between $0 \leq \varepsilon \leq 1$. When $\varepsilon = 0$, the algorithm never stops, while $\varepsilon = 1$ forces a stop irrespective of the values of the weights.

In addition to ε, there are also two other user-defined variables in the selection algorithm, including the constant C and the interval time τ. While the physical interpretation of τ is straightforward, C represents a pseudo diffusion constant. One may think of the selection algorithm as an induced diffusion in the reaction coordinate space; The larger the value of C, the faster the system will diffuse along the reaction coordinate space. Therefore C determines the strength or aggressiveness of the resampling algorithm. The most efficient value for C is dependent on the nature of the collective variable and the shape of its density ρ. Since the best choice of C for a given problem is somewhat of an art, we refer the interested reader to the ABMD tutorials on the AMBER webpage for insight into choosing this variable. Finally, we also note that the multiple walker selection mechanism can be invoked as is or in conjunction with the WT-ABMD for enhanced stability and convergence.

If any of these modules prove to be useful, please consider quoting the following papers: V. Babin, C. Roland and C. Sagui, *"Adaptively biased molecular dynamics for free energy calculations"*, J. Chem. Phys. **128**, 134101 (2008); V. Babin, V. Karpusenka, M. Moradi, C. Roland and C. Sagui, *"Adaptively biased molecular dynamics: an umbrella sampling method with a time-dependent potential"*, Int. J. Quant. Chem. **109**, 3666 (2009).

From Amber16, we implement these modules from SANDER to PMEMD and the modules are GPU compatible. To keep the consistency in format, we do a series of changes and updates to the usage of these modules. One big change is that you must set ***infe*** = 1 in &cntrl to activate these modules. Also, the input format has been changed to namelist style and reaction coordinate variables will be read from separate files. For the details, please read Subsection 21.6.7

infe This variable controls the usage of the non-equilibrium free energy method. When *infe=0*, the ABMD and related methods are turned off; when *infe=1*, they are turned on and the blocks `&smd`, `&pmd`, `&abmd` and `&bbmd` will be recognized. The default value is 0. Note that use of these algorithms may require a (minor) amount of post-processing by means of the nfe-umbrella-slice utility freely available in AmberTools described in Subsection 21.6.8.

21.6.2. Reaction Coordinates

A reaction coordinate is defined in the `colvar` namelist in a separate file. (see Fig. 21.4). This section must contain a `cv_type` keyword along with a value of type `STRING` and a list of integers `cv_i` (the number of integers

```
&colvar
   cv_type = STRING
   cv_ni = N, cv_nr = M
   cv_i = i1, i2, ..., iN
   cv_r = r1, r2, ..., rM
/
```

Figure 21.4.: *Syntax of reaction coordinate definition:* `cv_type` *is a* `STRING`*,* `cv_i` *is a list of integer numbers and* `cv_r` *is a list of real numbers.*

is defined by `cv_ni`). For some types of reaction coordinates the `colvar` section must also contain a list of real numbers, `cv_r`, whose length is defined by `cv_nr`.

The following reaction coordinates (specfied by `cv_type`) are currently implemented:

DISTANCE: distance (in Å) between two atoms whose indexes are read from the list `cv_i`.

COM_DISTANCE: distance between the center of mass of two atom groups. The `cv_i` list is interpreted as a list of indexes of participating atoms. Zeros separate the groups, the last zero is optional. eg: `cv_i = a1,...,aN,0,b1,...,bM,0`.

DF_COM_DISTANCE: difference of distances between the center of mass of first two atom groups and second two atom groups. The `cv_i` list is interpreted as a list of indexes of participating atoms. Zeros separate the groups, the last zero is optional. eg: `cv_i = a1,...,aN,0,b1,...bM,0,c1,...cL,0,d1,...,dK,0`, `DF_COM_DISTANCE` is `COM_DISTANCE(a1,...,aN,0,b1,...,bM) - COM_DISTANCE(c1,...cL,0,d1,...,dK)`.

LCOD: linear combination of distances (in Å) between pairs of atoms listed in `cv_i` with the coefficients read from `cv_r` list. For example, `i = 1,2,3,4` and `r = 1.0,-1.0` define the difference between 1-2 and 3-4 distances, *i.e.* LCOD = r1*distance(1, 2) + r2*distance(3,4).

ANGLE: angle (in radians) between the lines joining atoms with indexes `i1` and `i2` and atoms with indexes `i2` and `i3`.

COM_ANGLE: angle (in radians) formed by the center of mass of three atom groups. The `cv_i` list is interpreted as a list of indexes of participating atoms. Zeros separate the groups, the last zero is optional. eg: `cv_i = a1,...,aN,0,b1,...bM,0,c1,...,cK,0`.

TORSION: dihedral angle (in radians) formed by atoms with indexes `i1`, `i2`, `i3` and `i4`.

COM_TORSION: dihedral angle (in radians) formed by the center of mass of four atom groups. The `cv_i` list is interpreted as a list of indexes of participating atoms. Zeros separate the groups, the last zero is optional. eg: `cv_i = a1,...,aN,0,b1,...bM,0,c1,...,CK,0,d1,...,dL,0`.

COS_OF_DIHEDRAL: sum of cosines of dihedral angles formed by atoms with indexes in the list `cv_i`. The number of atoms must be a multiple of four.

SIN_OF_DIHEDRAL: sum of sines of dihedral angles formed by atoms with indexes in the list `cv_i`. The number of atoms must be a multiple of four.

PAIR_DIHEDRAL: sum of cosines of a list of angles each formed by summing two neighboring dihedral angles from a list formed by atoms with indices `cv_i`. The number of atoms must be a multiple of four. For a list of dihedral angles such as $\{\alpha_1, \ldots, \alpha_N\}$, PAIR_DIHEDRAL is $\sum_{i=1}^{N-1} cos(\alpha_i + \alpha_{i+1})$ which ranges between $-N+1$ and $N-1$.

PATTERN_DIHEDRAL: a particular pattern-recognizing function defined on a list of dihedral angles formed by atoms with indices `cv_i`. The number of atoms must be a multiple of four. The definition is particularly relevant for the dihedral angles with a binary-like behavior of being either around 0 or 180 (e.g., ω backbone dihedral angle). For a list of dihedral angles such as $\{\alpha_1, \ . \ . \ . \ ,\alpha_N \ \}$, PATTERN_DIHEDRAL is $\sum_{i=1}^{N} cos^2(\alpha_i/2)2^{i-1}$ which ranges between 0 and $2^N - 1$.

R_OF_GYRATION: radius of gyration (in Å) of atoms with indexes given in the `cv_i` list (mass weighted).

```
&colvar
   cv_type = 'MULTI_RMSD'
   cv_ni = 9, cv_nr = 21,
   cv_i = 1, 2, 3, 4, 0, 3, 4, 5, 0 ! the last zero is optional
   cv_r = 1.0, 1.0, 1.0, ! group #1, atom 1
        2.0, 2.0, 2.0, ! group #1, atom 2
        3.0, 3.0, 3.0, ! group #1, atom 3
        4.0, 4.0, 4.0, ! group #1, atom 4
       23.0, 23.0, 23.0, ! group #2, atom 3
          4.0, 4.0, 4.0, ! group #2, atom 4
          5.0, 5.0, 5.0 ! group #2, atom 5
/
```

Figure 21.5.: *An example of* `MULTI_RMSD` *variable definition.*

MULTI_RMSD: `RMS` (in Å, mass weighted) of `RMSD`s of several groups of atoms w.r.t. reference positions provided in the `cv_r` list. The `cv_i` list is interpreted as a list of indexes of participating atoms. Zeros separate the groups. An atom may enter several groups simultaneously. The `cv_r` array is expected to contain the reference positions (without zero sentinels). The implementation uses the method (and the code) introduced in Ref. [474]. An example of variable of this type is presented in Fig. 21.5. Two groups are defined here: one comprises the atoms with indexes 1, 2, 3, 4 (line 3 in Fig. 21.5, numbers prior to the first zero) and another one of atoms with indexes 3, 4, 5. The code will first compute the (mass weighted) `RMSD` (R_1) of atoms belonging to the first group w.r.t. reference coordinates provided in the `cv_r` array (first $12 = 4 \times 3$ real numbers of it; lines 4, 5, 6, 7 in Fig. 21.5). Next, the (mass weighted) `RMSD` (R_2) of atoms of the second group w.r.t. the corresponding reference coordinates (last $9 = 3 \times 3$ elements of the `cv_r` array in Fig. 21.5) will be computed. Finally, the code will compute the value of the variable as follows:

$$\text{value} = \sqrt{\frac{M_1}{M_1 + M_2} R_1^2 + \frac{M_2}{M_1 + M_2} R_2^2},$$

where M_1 and M_2 are the total masses of atoms in the corresponding groups.

N_OF_BONDS:

$$\text{value} = \sum_p \frac{1 - \left(r_p \big/ r_0\right)^6}{1 - \left(r_p \big/ r_0\right)^{12}},$$

where the sum runs over pairs of atoms p, r_p denotes distance between the atoms of pair p and r_0 is a parameter measured in Å. The `cv_r` array must contain exactly one element that is interpreted as r_0. The `cv_i` array is expected to contain pairs of indexes of participating atoms. For example, if 1 and 2 are the indexes of Oxygen atoms and 3, 4, 5 are the indexes of Hydrogen atoms and one intents to count all possible O-H bonds, the `cv_i` list must be (1, 3, 1, 4, 1, 5, 2, 3, 2, 4, 2, 5), that is, it must explicitly list all the pairs to be counted.

```
&colvar
   cv_type = 'N_OF_STRUCTURES'
   cv_ni = 9, cv_nr = 23,
   cv_i = 1, 2, 3, 4, 0, 3, 4, 5, 0 ! the last zero is optional
   cv_r = 1.0, 1.0, 1.0, ! group #1, atom 1
        2.0, 2.0, 2.0, ! group #1, atom 2
        3.0, 3.0, 3.0, ! group #1, atom 3
        4.0, 4.0, 4.0, ! group #1, atom 4
        1.0,           ! R0 for group #1
     23.0, 23.0, 23.0, ! group #2, atom 3
        4.0, 4.0, 4.0, ! group #2, atom 4
        5.0, 5.0, 5.0, ! group #2, atom 5
        2.0           ! R0 for group #2
/
```

Figure 21.6.: *An example of* `N_OF_STRUCTURES` *variable.*

HANDEDNESS:

$$\text{value} = \sum_a \frac{\mathbf{u}_{a,3} \cdot [\mathbf{u}_{a,1} \times \mathbf{u}_{a,2}]}{|\mathbf{u}_{a,1}|\,|\mathbf{u}_{a,2}|\,|\mathbf{u}_{a,3}|}\,,$$

where

$$\begin{aligned}
\mathbf{u}_{a,1} &= \mathbf{r}_{a+1} - \mathbf{r}_a \\
\mathbf{u}_{a,2} &= \mathbf{r}_{a+3} - \mathbf{r}_{a+2} \\
\mathbf{u}_{a,3} &= (1-w)(\mathbf{r}_{a+2} - \mathbf{r}_{a+1}) + w(\mathbf{r}_{a+3} - \mathbf{r}_a)\,,
\end{aligned}$$

and $\mathbf{r}_a$ denote the positions of participating atoms. The `cv_i` array is supposed to contain indexes of the atoms and the `cv_r` array may provide the value of w ($0 \le w \le 1$, the default is zero).

N_OF_STRUCTURES:

$$\text{value} = \sum_g \frac{1 - \left(R_g \middle/ R_{0,g}\right)^6}{1 - \left(R_g \middle/ R_{0,g}\right)^{12}}\,,$$

where the sum runs over groups of atoms, R_g denotes the `RMSD` of the group g w.r.t. some reference coordinates and $R_{0,g}$ are positive parameters measured in Å. The `cv_i` array is expected to contain indexes of participating atoms with zeros separating different groups. The elements of the `cv_r` array are interpreted as the reference coordinates of the first group followed by their corresponding R_0; then followed by the reference coordinates of the atoms of the second group, followed by the second R_0, and so forth. To make the presentation clearer, let us consider the example presented in Fig. 21.6. The atomic groups and reference coordinates are the same as the ones shown in Fig. 21.5. Lines 7 and 11 in Fig. 21.6 contain additional entries that set the values of the threshold distances $R_{0,1}$ and $R_{0,2}$. To compute the variable, the code first computes the mass weighted `RMSD` values R_1 and R_2 for both groups –much like in the `MULTI_RMSD` case– and then combines those in a manner similar to that used in the `N_OF_BONDS` variable.

$$\text{value} = \frac{1 - \left(R_1 \middle/ R_{0,1}\right)^6}{1 - \left(R_1 \middle/ R_{0,1}\right)^{12}} + \frac{1 - \left(R_2 \middle/ R_{0,2}\right)^6}{1 - \left(R_2 \middle/ R_{0,2}\right)^{12}}\,.$$

In other words, the variable “counts” the number of structures that match (stay close in `RMSD` sense) with the reference structures.

21.6.3. Steered Molecular Dynamics

The `&smd` namelist, if present in the `MDIN` file, activates the steered `MD` code (the method itself is extensively described in the literature: see for example Ref. [475] and references therein). The prefix NFE appears in several switches to do with steered MD: this stands for "Non-equilibrium Free Energy".

The following is recognized within the `&smd` namelist:

output_file sets the output file name. Default is '`nfe-smd.txt`'.

output_freq sets the output frequency (in `MD` steps). Default is 50.

cv_file sets the collective variable file name. Default is '`nfe-smd-cv`'.

There must be at least one reaction coordinate defined (that is, there must be at least one `&colvar` namelist in the `cv_file`). The steered `MD` code requires that additional entries be present in the `&colvar` namelist:

path the steering path whose elements must be real numbers. The `path` must include at least two elements. The upper limit on the number of entries is 20000. The elements define Catmull-Rom spline used for steering.

npath sets the number of elements in `path`. Default is 0.

path_mode The way steering paths are constructed. There are two modes available. In `SPLINE` mode (default) the path is approximated by spline that passes through the given points; in `LINES` mode the path is represented by the line segments joining the control points.

harm specifies the harmonic constant. If a single number is provided, e.g., `harm = 10.0`, then it is constant throughout the run. If two or more numbers are provided, e.g., `harm = 10.0, 20.0`, then the harmonic constant follows Catmull-Rom spline built upon the provided values.

nharm sets the number of elements in `harm`. Default is 0.

harm_mode The way harmonical paths are constructed, similar with `path_mode`.

An example of `MDIN` file and `CV.IN` file for steered `MD` is shown in Fig. 21.7. The reaction coordinate is defined in `cv.in`. The spring constant is set constant throughout the run and the steering path is configured from 5.0 to 3.0. The values of the reaction coordinate, harmonic constant and the work performed on the system are requested to be dumped to the `smd.txt` file every 50 MD steps.

21.6.4. Umbrella sampling

To activate the umbrella sampling code, the `&pmd` namlist must be present in the `MDIN` file. This is for SANDER only: `&pmd` is currently not activated for PMEMD, but will be available in the next AMBER release. Users who do want to use this code with PMEMD should write to *sagui@ncsu.edu* or *fpan3@ncsu.edu*. `The output_file`, `output_freq` and `cv_file` entries are recognized just as in the steered `MD` case presented earlier. The `cv_file` must contain at least one `&colvar` namelist section. For umbrella sampling, the `&colvar` section(s) must contain two additional entries:

anchor_position: this consists of four real numbers $(r1, r2, r3, r4)$ that determine the rectangle of the umbrella (harmonic) potential. The default value is that all of the $r's$ is set to zero.

anchor_strength: two non-negative real numbers $(k1, k2)$ that set the harmonic constant for the umbrella (harmonic) potential. The default value is zero.

The umbrella (harmonic) potential U is determined by (supposing R is the value of reaction coordinate)

- $U = k1 * (r1 - r2) * R \quad (R \leqslant r1)$
- $U = 0.5 * k1 * (R - r2)^2 \quad (r1 < R \leqslant r2)$

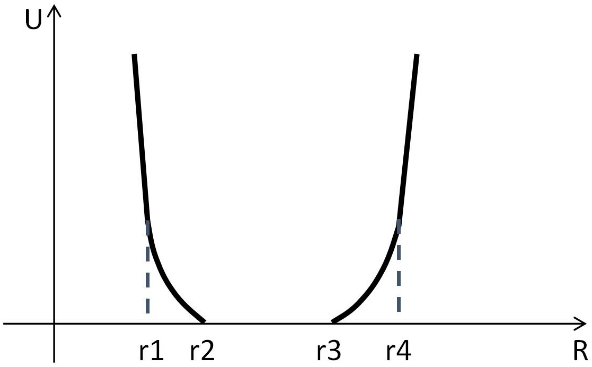
```
title line
&cntrl
..., infe = 1
/

&smd
   output_file = 'smd.txt'
   output_freq = 50
   cv_file = 'cv.in'
/
```

```
cv_file
&colvar
  cv_type = 'DISTANCE'
  cv_ni = 2
  cv_i = 5, 9
  npath = 2, path = 5.0, 3.0, path_mode = 'LINES',
  nharm = 1, harm = 10.0
/
```

Figure 21.7.: *An example `MDIN` file and `CV.IN` file for steered `MD`. Only the relevant part is shown.*

- $U = 0 \qquad (r2 < R \leqslant r3)$
- $U = 0.5 * k2 * (R - r3)^2 \quad (r3 < R \leqslant r4)$
- $U = k2 * (r4 - r3) * R \quad (R > r4)$

A plot of the umbrella potential is shown below



eg1: if $r2 = r3$, $r1 \ll r2$ and $r4 \gg r3$, then the generated U is simply the traditional harmonic potential.

eg2: if $r1$ is slightly less than $r2$ and $r4$ is slightly larger than $r3$, also with very large $k1$, $k2$, the reaction coordinate is restrained in the range ($r2$, $r3$) with no potential added.

An example of an `MDIN` file and `CV.IN` file for an umbrella sampling simulation is shown in Fig. 21.8. The first reaction coordinate here is the angle formed by the lines joining the 5th with 9th and 9th with 15th atoms. It is to be harmonically restrained near 1.0 *rad* (`anchor_position` entry) using the spring of strength 10.0 $kcal/mol/rad^2$ (`anchor_strength` entry). The second reaction coordinate requested in Fig. 21.8 is a dihedral angle (`type = 'TORSION'`) formed by the 1st, 2nd, 3rd and 4th atoms (the `cv_i` array). It is to be restrained near zero with strength 23.8 $kcal/mol/rad^2$. The values of the reaction coordinate(s) are to be dumped every 50 `MD` steps to the `pmd.txt` file. Another example of restraining reaction coordinate in a specific range is shown in Fig. 21.9.

The reaction coordinates here are φ and ψ angles of dialanine. φ is restrained between -2.0 *rad* and 2.0 *rad*, ψ is restrained between -1.8 *rad* and 1.8 *rad*.

The `NFE` implementation of umbrella sampling works correctly with the Amber standard replica-exchange MD described earlier in this manual. It assumes, however, that the number and type of reaction coordinate(s) are the same in all replicas. On the other hand, both `anchor_position` and `anchor_strength` may be different for different temperatures. For replica-exchange MD the output files (set by the `output_file` keyword on a per-replica basis) are temperature bound (or `MDIN`-bound, since there is one-to-one temperature-`MDIN` correspondence).

```
title line
&cntrl
..., infe = 1
/

&pmd
   output_file = 'pmd.txt'
   output_freq = 50
   cv_file = 'cv.in'
/
```

```
cv_file
&colvar ! first
   cv_type = 'ANGLE'
   cv_ni = 3, cv_i = 5, 9, 15
   anchor_position = -10.0,1.0,1.0,10.0
   anchor_strength = 10.0,10.0
/
&colvar ! second
   cv_type = 'TORSION'
   cv_ni = 4, cv_i = 1, 2, 3, 4
   anchor_position = -10.0,0.0,0.0,10.0
   anchor_strength = 23.8,23.8
/
```

Figure 21.8.: *An example `MDIN` file and `CV.IN` file for umbrella sampling (only relevant part is presented in full).*

```
cv_file
&colvar ! phi
   cv_type = 'TORSION'
   cv_ni = 4, cv_i = 5, 7, 9, 15
   anchor_position = -2.05,-2.0,2.0,2.05
   anchor_strength = 500.0,500.0
/
&colvar ! psi
   cv_type = 'TORSION'
   cv_ni = 4, cv_i = 7, 9, 15, 17
   anchor_position = -1.85,-1.8,1.8,1.85
   anchor_strength = 500.0,500.0
/
```

Figure 21.9.: *An example `CV.IN` file to restrain the φ and ψ of dialanine.*

21.6.5. Adaptively Biased Molecular Dynamics

The implementation has a very simple and intuitive interface: the code is activated if either an `&abmd` (both SANDER and PMEMD) or an `&bbmd` (SANDER only) namelist is present in the `MDIN` file (the difference between those "flavors" is purely technical and will become clear later). Unlike in the `&smd` and `&pmd` cases, the dimensionality of a reaction coordinate (the number of `&colvar` namelists in the `cv_file`) cannot exceed five (though three is already hardly useful due to statistical reasons).

As previously noted, in order to activate the ABMD and related algorithm, the variable ***infe*** in &cntrl must be set to unity (*i.e.* ***infe*** = 1; default value ***infe*** = 0).

In addition to the `cv_file` entry, the following entries are recognized within the `&abmd` (or `&bbmd`) namelist:

mode sets the execution mode. There are three modes available: `'ANALYSIS' | 'UMBRELLA' | 'FLOODING'`. In `ANALYSIS` mode the dynamics is not altered. The only effect of this mode is that the value(s) of the reaction coordinate(s) is(are) dumped every `monitor_freq` to `monitor_file`. In `UMBRELLA` mode, biasing potential from the `umbrella_file` is used to bias the simulation ($\tau_F = \infty$, biasing potential does not change). In `FLOODING` mode the adaptive biasing is enabled.

monitor_file sets the name of the file to which value(s) of reaction coordinate(s) (along with the magnitude of biasing potential in `FLOODING` mode) are dumped.

monitor_freq the frequency of the output to the `monitor_file`.

timescale τ_F, the flooding timescale in picoseconds (only required in `FLOODING` mode).

umbrella_file biasing potential file name (the file must exist for the `UMBRELLA` mode).

In `FLOODING` mode, the following two entries are optional:

snapshots_basename sets the name of the file to which the biasing potential is dumped during the simulation for snapshot.

snapshots_freq the frequency of dumping snapshot biasing potential (in MD steps). If `snapshots_freq` is not specified, the snapshot biasing potential will not be dumped.

and the `&colvar` namelist for `&abmd` method must also contain the following entries:

cv_min smallest desired value of the reaction coordinate (required, unless the reaction coordinate is limited from below).

cv_max largest desired value of the reaction coordinate (required, unless the reaction coordinate is limited from above).

resolution the "spatial" resolution for the reaction coordinate.

To access the biasing potential files created in the course of `FLOODING` simulations, the `nfe-umbrella-slice` utility is provided (it prints a short description of itself if invoked with `--help` option).

The multiple-walker selection algorithm can improve the simulation by resampling between different walkers. The well-tempered ABMD can lead to a smoother convergence to the desired free energy. These two algorithm are implemented to SANDER and PMEMD from Amber16 onwards.

The multiple-walker selection algorithm currently works with `&abmd` only. The algorithm should be used only within the multiple-walker scheme (*i.e.*, when command-line **-rem** flag is set to zero). The following entries are recognized regarding with the selection algorithm (selection algorithm can work with `FLOODING` and `UMBRELLA` mode):

selection_freq positive integer number that sets the frequency of the resampling algorithm (in MD steps). If `selection_freq` is not specified, the selection algorithm will not be used.

selection_constant positive real number that sets the parameter C. if `selection_freq` is specified, specifying `selection_constant` is required (no default value). Parameter C is to determine how strong the selection mechanism is. If C is too large, all the walkers will be replaced with the most dominant one. If C is too small, there will be no killing/duplicating of walkers.

selection_epsilon positive real number (typically less than unity) that sets the stopping criterion parameter ε . Parameter ε determines the threshold for stopping the selection algorithm. If `selection_epsilon` is not specified, there will be no stop to the algorithm. If `selection_epsilon` is equal or larger than one, the algorithm will be stopped after the first attempt.

The well-tempered flavor can be used within either `&abmd` or `&bbmd` namelist. There are two entries relevant to the well-tempered feature:

wt_temperature positive real number that sets the pseudo-temperature T’. If this flag is not specified, conventional ABMD will be used (*i.e.*, T’ → ∞ or β’ → 0). The smaller the T’; the smoother/slower the convergence.

wt_umbrella_file the file name of true biasing potential after modification by 1 + (T /T’) in which T is the reference temperature of the system (temp0).

An example `MDIN` file and `CV.IN` file for the `&abmd` flavor of `ABMD` is shown in the Fig. 21.10.

```
title line
&cntrl
..., infe = 1
/

&abmd
   mode = 'FLOODING'

   monitor_file = 'abmd.txt'
   monitor_freq = 33
   cv_file = 'cv.in'

   umbrella_file = 'umbrella.nc'

   timescale = 100.0 ! in ps

   selection_freq = 10000
   selection_constant = 0.001

   wt_temperature = 10000.0
   wt_umbrella_file = 'wt_umbrella.nc'
/
```

```
cv_file
&colvar
  cv_type = 'DISTANCE'
  cv_ni = 2, cv_i = 5, 9
  cv_min = -1.0, cv_max = 10.0 ! min is not needed for DISTANCE
  resolution = 0.5 ! required for mode = FLOODING
/
```

Figure 21.10.: *An example `MDIN` file and `CV.IN` file for `ABMD` (only the relevant part is presented in full).*

In this example, the reaction coordinate is defined as the distance between the 5th and 9th atoms (more than one reaction coordinates might be requested by mere inclusion of additional `&colvar` subsections). The `mode` is set to `FLOODING` thus enabling the adaptive biasing with flooding timescale $\tau_F = 100ps$. The region of interest of the reaction coordinate is specified to be between -1 Å and 10 Å and the resolution is set to 0.5 Å. The lower bound (-1 Å) could have been omitted for `DISTANCE` variable: the default value of zero would be used in such case. The code will try to load the biasing potential from the `umbrella.nc` file and use it as the value of $U(t|\xi)$ at the beginning of the run. The biasing potential built in the course of simulation will be saved to the same file (`umbrella.nc`) every time the `RESTRT` file is written. The selection algorithm is used with the frequency of selection defined as 10000 MD steps and selection constant defined as 0.001. The well-tempered algorithm is also used, with the pseudo-temperature defined as 10000 K in and the true biasing potential will be dumped as `wt_umbrella.nc` file. The `nfe-umbrella-slice` utility can then be used to access its content. An `MDIN` file for the follow up biased run at equilibrium would look much like the one shown in the Fig. 21.10, but with `mode` changed from `FLOODING` to `UMBRELLA`.

For SANDER only, the `&abmd` code works correctly with replica-exchange (that is, for `-rem` flag set to 1). In such case the monitor and umbrella files are temperature-bound (unlike, e.g., `MDOUT` and `MDCRD` files that require post processing). If number of `sander` groups exceeds one (the flag `-ng` is greater than one) and `-rem` flag is set to zero, the code runs *multiple walkers* `ABMD`. In both cases the number and type(s) of variable(s) must be the same across all replicas.

Finally, the `&bbmd` flavor allows one to run replica-exchange (`AB`)`MD` with different reaction coordinates and different modes (`ANALYSIS`, `UMBRELLA` or `FLOODING`) in different replicas (along with different temperatures, if desired). To this end, the `-rem` flag must be set to zero and the `&bbmd` sections must be present in all `MDIN` files. The `MDIN` file for the replica of rank zero (first line in the group file) is expected to contain additional information as compared to `&abmd` case (an example of such `MDIN` file for replica zero is shown in Fig. 21.11). The `MDIN` files for all other replicas except zero do not need any additional information, and therefore take the same form as in the `&abmd` flavor (except that the namelist is changed from `&abmd` to `&bbmd`, thus activating a slightly different code path). Each `MDIN` file may define its own reaction coordinates, have different `mode` and temperature if desired.

Within the first replica `&bbmd` namelist the following additional entries are recognized:

exchange_freq number of `MD` steps between the exchange attempts.

exchange_log_file the name of the file to which exchange statistics is to be reported.

exchange_log_freq frequency of `exchange_log_file` updates.

mt19937_seed seed for the random generator (Mersenne twister [476]).

mt19937_file the name of the file to which the state of the Mersenne twister is dumped periodically (for restarts).

The `MDOUT`, `MDCRD`, `RESTRT`, `umbrella_file` and `monitor_file` files are `MDIN`-bound in course of the `bbmd`-enabled run. An example that uses this kind of replica exchange is presented in Ref.457.

21.6.6. Swarms-of-Trajectories String Method

ABMD is a robust method for calculating free energy landscapes as a function of a small number of collective variables. Since the required computer time grows enormously with the number of collective variables, ABMD is best for exploring one- or two-dimensional phase spaces. However, rather than calculating full n-dimensional free energy maps, it is often fruitful to focus on the so-called Minimum Free Energy Path (MFEP) which the system is likely to take when transitioning between two minima. Calculating a MFEP in a complicated phase space is often difficult, and so-called ”string methods"[477][471] represent one of the best approaches for finding the MFEP. Since sampling in string methods is essentially limited to regions around the MFEP, the cost of the method scales linearly with the length of the string or path, but only weakly on the number of collective variables. This results in considerable computational savings since the full free energy landscape is not calculated.

The swarms-of-trajectories string method (STSM)[471] is one of the most popular versions of the string method and has been implemented here by Dr. Moradi (moradi@uark.edu) for SANDER only (for a PMEMD version of the code, please write to fpan3@ncsu.edu or sagui@ncsu.edu). It is a path-finding algorithm that refines a putative transition pathway iteratively until the path is deemed to have been converged. The string is defined by a number of nodes or images parameterized in a high-dimensional space of collective variables, whose position is updated

```
title line
&cntrl
..., infe = 1
/

&bbmd

   ! 0th replica only

   exchange_freq = 100 ! try for exchange every 100 steps

   exchange_log_file = 'bbmd.log'
   exchange_log_freq = 25

   mt19937_seed = 123455 ! random generator seed
   mt19937_file = 'mt19937.nc' ! file to store/load the PRG

   ! not specific for 0th replica

   mode = 'ANALYSIS'

   monitorfile = 'bbmd.01.txt' ! it is wise to have different
                               ! names in different replicas
   monitor_freq = 123
   cv_file = 'cv.in'
/
```

```
cv_file
&colvar
  cv_type = 'DISTANCE'
  cv_ni = 2, cv_i = 5, 9
/
```

Figure 21.11.: *An example `MDIN` file and `CV.IN` file for `&bbmd` flavor of `ABMD` (only the relevant part is presented in full)(for SANDER only).*

iteratively. The center of each image is first used as a restraining center to generate representative conformations at the current center before allowing of a small change in this center for the next iteration. The change in the center of each image is estimated by averaging over the drifts of a swarm of short unbiased trajectories all starting at the current image position (generated using the constrained simulations. Thus, each iteration consists of a series of restrained and free simulations. In the current serial version of the code, these simulations are performed independently. In parallel versions -- which are more efficient -- a very large number of replicas is required which are run in parallel; this method is particularly efficient on large supercomputers.

To invoke the swarms-of-trajectories string method, the `&stsm` must be invoked in the `MDIN` file. For a string consisting of N_s nodes each requiring M copies $N_s \times M$ replicas will be required. The parallel implemention of the STSM method is based on iterative restrained and free MD simulations followed by a reparametrization of the image centers defined in a multidimensional collective variable space ξ. For the i^{th} iteration, first M copies of the n^{th} image are generated around the old center ξ_n^{i-1} by MD equilibration lasting τ_E timesteps. The generated M copies of the n^{th} image are expected to be close to ξ_n^{i-1} for time τ_E, assuming that the invoked harmonic constant k for the restraining potential is large enough. The parameters τ_E and k thus need to be appropriately chosen in order to ensure that all copies of each image will be close to the image center. The restraint is then released, and each copy (swarm) is allowed to drift for τ_R timesteps. The newly shifted center ξ_n^i for the n^{th} image is then determined by averaging over all drifted copies $\xi_{n,m}^t$ at time $t = \tau_E + \tau_R$. The resulting string of images is then smoothed using a linear interpolation protocol. A smoothing parameter ε with $0 \leq \varepsilon \leq 1$ determines the smoothness of the curve; it is recommended that ε be of the order of $1/(N_s - 1)$. The last setep is a reparameterization, which gain follows a linear interpolation protocol in order to generate N_s equidistant centers along the string. The two key parameters of the method are M and τ_R. Generally, the large the M and the shorter τ_R, the smoother (but slower) the evolution of the MFEP will be. These variables must be optimized empirically, but typically 10 - 30 copies and 5 - 20 ps are reasonable values. It is often advantageous to set $\tau_E = \tau_R$.

An improved sequential repeat version of the algorithm has also been implemented, which avoids the large number of copies and does not require a large number of processors to run. Here a new variable N_R is introduced, as the number of repeat runs for each replica. Now for each copy, it will run around the old center ξ_n^{i-1} for N_R times sequentially. And each repeat run can be equally considered as a parallel run of a new copy around the old center. Namely, the new shifted center will be determined by averaging on $N_R \times M$ copies. So the number of processors needed will be reduced to $1/N_R$, while the running time will be multipled by N_R.

The following is recognized within the &stsm namelist:

image positive integer number that sets the image id (between 1 and N). Default is 0.

repeats positive integer number that sets the number of repeat runs, should be the same for each image and each copy. Equal to parallel implementation when not set. Default is 1.

equilibration non-negative integer number that sets the number of MD steps specified for biased equlibration (restraining) at each iteration. Default is 0.

release Number of MD steps specified for the release (drift) at each iteration. Note: the total number of iterations is determined by the total simulation time (`nstlim` flag in `mdin` file) divided by total time for each iteration given by `equilibration+release`.

smoothing positive number that sets the smoothing parameter for reparametrization (between 0 and 1). Smoothing parameter should be, preferably, on the order of $1/(N_s - 1)$. If this flag is not used, no smoothing will be performed.

report_centers a string that determines if drifted and/or smoothed and/or reparametrized centers will be reported. The default value is `NONE` and other available options include `ALL`,`DRIFT`,`SMOOTHED`, `REPARAMETRIZED`, `NO_DRIFT`, `NO_SMOOTHED`, `NO_REPARAMETRIZED`.

`The output_file`, `output_freq` and `cv_file` entries are recognized just as `&smd` and `&pmd`, the information of reaction coordinates will be read from `cv_file`. The number of collective variables can not exceed five. (here be attention that the `anchor_postion` and `anchor_strength` will be defined using the traditional harmonical potential, different with `&pmd`!). An example of `MDIN` file and `CV.IN` file for STSM in parallel case is shown in Fig. 21.12. Here we run 8 images along the path, with `I` defining the image ID. We run 980 MD steps for equlibration and 20 MD steps release at each iteration, so there are totally 1000 MD steps for each iteration. With nstlim set to 10000, 10 iterations will be carried out. The smoothing parameter is set to 0.1 and all the centers will be reported. For each image, 16 copies will be run in parallel, with `J` defining the copy ID. The evolution of reaction coordinate will recorded in the file `stsm.00I.J.txt`. For this run, at least 128(8×16) processors are needed. Another example of `MDIN` file of equivalent sampling level in sequential case is shown in Fig. 21.13. Here we still have 8 images to run. We set the number of repeats to be 16, namely 16 repeat runs for each image to get the new drifted center. Therefore, 16000 MD steps are needed for one iteration, and so we set nstlim to 160000 to complete 10 iterations. For this run, 8 processors are needed at least.

Part of sample `MDOUT` file is shown in Fig. 21.14. The restoring restraint part will be only in sequential run, since the restraint needs to be restored after each repeat. The values of reaction coordinates before reporting centers are the averaged value over repeats for this copy and the instantaneous value. All the centers will be reported only in the `MDOUT` file of first copy of first image. The drifted centers are the averaged value over copies, and also the smoothed ans reparametrized centers can be reported. Always the reparametrized centers will be extracted to draw the MFEP in the phase space.

```
title line
&cntrl
..., nstlim = 10000
..., infe = 1
/

&stsm      ! parallel case, I from 1 to 8, J from 1 to 16
  image = I
  equilibration = 980
  release = 20
  smoothing = 0.1
  report_centers = 'ALL'

  output_file = 'stsm.00I.J.txt'
  output_freq = 10
  cv_file = 'cv.I'
/
```

```
cv_file
&colvar ! phi
   cv_type = 'TORSION'
   cv_ni = 4, cv_i = 5, 7, 9, 15
   anchor_position = -3.00
   anchor_strength = 20.0
/
&colvar ! psi
   cv_type = 'TORSION'
   cv_ni = 4, cv_i = 7, 9, 15, 17
   anchor_position = 3.00
   anchor_strength = 20.0
/
```

Figure 21.12.: *An example* `MDIN` *file and* `CV.IN` *file for* `&stsm` *in parallel case (only the relevant part is presented in full)*

21.6.7. Implementation in PMEMD

From Amber16 and onwards, the above features have been implemented in PMEMD. Becaused of GPU-enhanced calculations in PMEMD, users can now utilize GPUs to improve the speed of free energy sampling a lot with *pmemd.cuda* and *pmem.cuda.MPI*. This is very important when we want to study system in explicit solvent. With *pmemd* and *pmemd.MPI* the option to use CPUs is also kept, and the whole method has been tested sucessfully on both implicit and explicit enviroment. If you have questions with regards to the PMEMD implementation of ABMD and related algorithms, please contact Feng Pan <fpan3@ncsu.edu> and Dr. M. Moradi <moradi@uark.edu>.

Not all functionality has been implemented from SANDER to PMEMD, and some functionality has been modified. Here are all the changes:

- Naming was changed, all the input format is changed to namelist style. The reaction coordinate (collective variable) information is read from a separate file so it can be reusable by different runs.

- A new variable ***infe*** is added within the namelist &cntrl, to control the usage of the NFE method in a friendly way. To disable the NFE method, set ***infe*** to 0; to enable it, set ***infe*** to 1.

- Temperature replica-exchange method in `&abmd` block is not available(that is `-rem` must be set to 0).

```
title line
&cntrl
..., nstlim = 160000
..., infe = 1
/

&stsm      ! sequential case, I from 1 to 8
  image = I
  repeats = 16
  equilibration = 980
  release = 20
  smoothing = 0.1
  report_centers = 'ALL'

  output_file = 'stsm.00I.txt'
  output_freq = 10
  cv_file = 'cv.I'
/
```

Figure 21.13.: *An example `MDIN` file for `&stsm` in sequential case (only the relevant part is presented in full)*

```
NFE : #   restoring restraint:
NFE : #   << colvar(1) = -3.000000 >>
NFE : #   << colvar(2) = 3.000000 >>
NFE : #   equilibration begins...
......
NFE : #   << colvar(1) = -2.500688 -2.586429 >>
NFE : #   << colvar(2) = 2.782725 3.082205 >>
NFE : #  drifted center of image 1 :            8      -2.54041796       2.70644813
NFE : #  drifted center of image 2 :            8      -2.54963153       2.71715138
......
NFE : #  drifted center of image 8 :            8       1.02191205       0.16837852
NFE : #  smoothed center of image 1 :            8      -2.54041796       2.70644813
NFE : #  smoothed center of image 2 :            8      -2.60416697       2.75924174
......
NFE : #  smoothed center of image 8 :            8       1.02191205       0.16837852
NFE : #  reparametrized center of image 1 :            8      -2.54041796       2.70644813
NFE : #  reparametrized center of image 2 :            8      -2.06027108       2.47738701
......
NFE : #  reparametrized center of image 8 :            8       1.02191205       0.16837852
```

Figure 21.14.: *An example of `MDOUT` file for STSM run (only part is presentd, and some centers are also omitted)*

- &pmd and &bbmd (for umbrella sampling and T-REMD or H-REMD) is not currently available with this release. PMEMD code (to be used “as is”) may be obtained by writing to sagui@ncsu.edu or fpan3@ncsu.edu. Code will be put on AMBER website via an upcoming tutorial.
- Two new entries have been added to the blocks `&abmd` and `&bbmd` to give snapshots of the biasing potential during the simulation:

  snapshots_basename = STRING : sets the snapshots file name.

  snapshots_freq = INTEGER : sets the snapshot frequency. (in MD steps)
- It is not sensible to run a one-replica simulation with multiple GPUs

- Two new algorithms have been added to both SANDER and PMEMD: (1) a selection algorithm for multiple-walker ABMD; (2) the well-tempered ABMD (WT-ABMD).
- The swarms-of-trajectories string method (STSM) has been added to SANDER. PMEMD code (GPU compatible) may be obtained by writing to sagui@ncsu.edu or fpan3@ncsu.edu.
- Several new reaction coordinates are added, which include
  - `type = SIN_OF_DIHEDRAL`
  - `type = PAIR_DIHEDRAL`
  - `type = PATTERN_DIHEDRAL`
  - `type = DF_COM_DISTANCE`

21.6.8. Post-processing of biasing potential

When you get the biasing potential (`*.nc` file), you can always use the `nfe-umbrella-slice` utility to access its content and get a friendly-written ASCII file from which one can obtain the free energy map. The output is the free energy value, which is the opposite of the biasing potential ($f = -U$ ($units\ kcal/mol$)). The `nfe-umbrella-slice` utility has been included in AmberTools.

Usage: `nfe-umbrella-slice [options] bias_potential.nc`

Options:

`-h,--help` Print out a usage summary

`-p,--pretend` Only print out the basic properties of source without biasing potential data (off by default)

`-g,--gradient` Print out the gradients (off by default)

`-t,--translate` Translate the numerical value of biasing potential by a real number (0 by default)

`-d,--dimensions` Set the way of slice in different dimensions. The format is “D1:D2:...:Dn”, n is the number of dimensions. Each D can only be set with one number or three numbers, seperated by comma. If only one number is set, the variable will be fixed at that value. If three numbers are set, the first two define the boundary of slice and the last one defines the number of points.

Example:

- `nfe-umbrella-slice -d "-5.0,5.0,50" 1d-bias.nc > FE.dat`

This processes the 1-dimensional biasing potential file `1d-bias.nc` and prints out the results to `FE.dat`. The variable will be taken from -5.0 to 5.0 using 50 points.

- `nfe-umbrella-slice -g -t 50.0 -d "1.0:-2.0,2.0,20" 2d-bias.nc > FE.dat`

This processes the 2-dimensional biasing potential file `2d-bias.nc` and prints out the results to `FE.dat`. All the free energy will be incremented by a constant 50.0. The gradients in both dimensions will be printed out. For the first dimension, the variable will be fixed at 1.0; for the second dimension, the variable will be taken from -2.0 to 2.0 using 20 points.

- `nfe-umbrella-slice wt_umbrella.nc > wt_FE.dat`

This processes the biasing potential after WT-ABMD and prints out the results to `wt_FE.dat`. The default dimensional information is obtained and used by the program from the biasing potential file.

21.7. Steered Molecular Dynamics (SMD) and the Jarzynski Relationship

21.7.1. Background

SMD applies an external force onto a physical system, and drives a change in coordinates within a certain time. Several applications have come from Klaus Schulten's group.[478] An implementation where the coordinate in question changes in time at constant velocity is coded in this version of Amber. The present implementation has been done by the group of Prof. Dario Estrin in Buenos Aires <dario@q1.fcen.uba.ar> by Marcelo Marti <marcelomarti@yahoo.com> and Alejandro Crespo <alec@qi.fcen.uba.ar>, and in the group of Prof. Adrian Roitberg at the University of Florida <roitberg@ufl.edu>.[479]

The method should be thought of as an umbrella sampling where the center of the restraint is time-dependent as in:

$$V_{rest}(t) = (1/2)k[x - x_0(t)]^2$$

where x could be a distance, an angle, or a torsion between atoms or groups of atoms.

This methodology can be used then to drive a physical process such as ion diffusion, conformational changes and many other applications. By integrating the force over time (or distance), a generalized work can be computed. This work can be used to compute free energy differences using the so-called Jarzynski relationship.[480–482] This method states that the free energy difference between two states A and B (differing in their values of the generalized coordinate x) can be calculated as

$$\exp(-\Delta G/k_B T) = \langle \exp(-W/k_B T) \rangle_A \tag{21.16}$$

This means that by computing the work between the two states in question, and averaging over the initial state, equilibrium free energies can be extracted from non-equilibrium calculations. In order to make use of this feature, SMD calculations should be done, with different starting coordinates taken from equilibrium simulations. This can be done by running sander multiple times, or by running *multisander* (Section 17.11). There are examples of the various modes of action under the *test/jar* directories in the Amber distribution.

21.7.2. Implementation and usage

To set up a SMD run, set the *jar* variable in the &cntrl namelist to 1. The change in coordinates is performed from a starting to an end value in *nstlim* steps.

To specify the type and conditions of the restraint an additional ".RST" file is used as in *nmropt=1*. (Note that *jar=1* internally sets *nmropt=1*.) The restraint file is similar to that of NMR restraints (see Section 23.1), but fewer parameters are required. For instance, the following RST file could be used:

```
# Change distance between atoms 485 and 134 from 15 A to 20 A
&rst iat=485,134, r2=15., rk2 = 5000., r2a=20. /
```

Note that only $r2, r2a$ and *rk2* are required; *rk3* and $r3$ are set equal to these so that the harmonic restraint is always symmetric, and $r1$ and $r4$ are internally set so that the restrain is always operative. An SMD run changing an angle, would use three *iat* entries, and one changing a torsion needs four. As in the case of NMR restraints, group inputs can also be used, using *iat<0* and defining the corresponding groups using the *igr* flag.

The output file differs substantially from that used in the case of nmr restraints. It contains 4 columns: $x_0(t)$, x, force, work. Here work is computed as the integrated force over distances (or angle, or torsion). These files can be used for later processing in order to obtain the free energy along the selected reaction coordinate using Jarzynski's equality.

Example

The following example changes the distance between two atoms along 1000 steps:

```
Sample pulling input
&cntrl
nstlim=1000, cut=99.0, igb=1, saltcon=0.1,
ntpr=100, ntwr=100000, ntt=3, gamma_ln=5.0,
ntx=5, irest=1, ig = 256251,
ntc=2, ntf=2, tol=0.000001,
dt=0.002, ntb=0, tempi=300., temp0=300.,
jar=1,
/
&wt type='DUMPFREQ', istep1=1, /
&wt type='END', /
DISANG=dist.RST
DUMPAVE=dist_vs_t
LISTIN=POUT
LISTOUT=POUT
```

Note that the flag *jar* is set to 1, and redirections to the *dist.RST* file are given. In this example the values in the output file *dist_vs_t* are written every *istep=1* steps.

The restraint file *dist.RST* in this example is:

```
# Change distance between atoms 485 and 134 from 15 A to 20.0 A
&rst iat=485,134, r2=15., rk2 = 5000., r2a=20.0, /
```

and the output *dist_vs_t* file might contain:

```
15.00000 15.12396 -1239.55482 0.00000
15.00500 14.75768 2470.68119 3.07782
15.01000 15.13490 -1246.46571 6.13835
15.01500 15.15041 -1350.03026 -0.35289
15.02000 14.77085 2481.56731 2.47596
15.02500 15.12423 -987.34073 6.21152
15.03000 15.18296 -1520.41603 -0.05787
15.03500 14.79016 2431.22399 2.21915
.......
19.97000 19.89329 4.60255 67.01305
19.97500 19.87926 4.78696 67.03652
19.98000 19.86629 4.54839 67.05986
19.98500 19.85980 3.75589 67.08062
19.99000 19.86077 2.58457 67.09647
19.99500 19.86732 1.27678 67.10612
```

In this example, the work of pulling from 15.0 to 20.0 (over 2 ps) was 67.1 kcal/mol. One would need to repeat this calculation many times, starting from different snapshots from an equilibrium trajectory constrained at the initial distance value. This could be done with a long MD or a REMD simulation, and postprocessing with ptraj to extract snapshots. Once the work is computed, it should be averaged using Eq. 21.16 to get the final estimate of the free energy difference. The number of simulations, the strenght of the constraint, and the rate of change are all important factors. The user should read the appropriate literature before using this method. It is recommended that the width of the work distribution do not exceed 5-10% for faster convergence. In many cases, umbrella sampling (see Section 21.4) may be a better way to estimate the free energy of a conformational change.

22. Constant pH calculations

A constant pH molecular dynamics method was developed by John Mongan for simulations run with the Generalized Born implicit solvent model [483] and Jason Swails for simulations with explicit solvent. [484] Using either constant pH method requires minor modifications to the process of generating the prmtop file and also requires a second input file describing the titrating residues.

22.1. Background

Traditionally, molecular dynamics simulations have employed constant protonation states for titratable residues. This approach has many drawbacks. First, assigning protonation states requires knowledge of pKa values for the protein's titratable groups. Second, if any of these pKa values are near the solvent pH there may be no single protonation state that adequately represents the ensemble of protonation states appropriate at that pH. Finally, since protonation states are constant, this approach decouples the dynamic dependence of pKa and protonation state on conformation.

The constant pH method implemented in *sander* and *pmemd* addresses these issues through Monte Carlo sampling of the Boltzmann distribution of protonation states concurrent with the molecular dynamics simulation. The protonation state distribution is affected by solvent pH, which is set as an external parameter. Residue protonation states are changed by changing the partial charges on the atoms of the protonable residue.

22.2. Preparing a system for constant pH

Amber provides definitions for titrating side chains of ASP, GLU, HIS, LYS, TYR, and CYS. See below if you need other titrating groups.

Begin by preparing your PDB file as you normally would for use with LEaP. Edit the PDB file, replacing all histidine residue names (HIS, HID, or HIE) with HIP. Change all ASP and ASH to AS4 and all GLU and GLH to GL4. The others—LYS, TYR, and CYS—have the same name. This ensures that the prmtop file will have a hydrogen defined at every possible point of protonation. Note that these changes should only be applied to residues that you wish to titrate.

Run LEaP with the *leaprc.constph* command file. This file loads all parameters that were used for the reference compounds. You can load this file with the following command:

```
source leaprc.constph
```

This loads the ff14SB force field. In addition, it loads the special carboxylate residue libraries and force field modifications—constph.lib and frcmod.constph—that defines a hydrogen atom at each protonable location (syn- and anti- for both oxygens) along with improper torsions to prevent them from rotating into each other. It also sets the GB solvation radii (PBradii) to mbondi2, which was the set used to parameterize the reference compounds. Now load your edited PDB file and proceed as usual to create the prmtop and prmcrd files. Changing any of the above parameters should be closely checked by titrating the reference compounds and ensuring the predicted pKa matches.

Once you have the prmtop file, you need to generate a cpin file. The cpin file describes which residues should titrate, and defines the possible protonation states and their relative energies. A python script, *cpinutil.py*, is provided to generate this file. It takes a prmtop file as input, on the command line along with the GB model you wish to evaluate protonation transitions in, and writes the cpin file to STDOUT. Here is an example of generating the cpin file from your prmtop file, 'prmtop' using the igb=2 GB model:

```
cpinutil.py -p prmtop -igb 2 > cpin
```

The *cpinutil.py* program accepts a number of flags that modify its behavior. By default, all residues start in protonation state 0: deprotonated for ASP and GLU, protonated for LYS, TYR, and CYS, and doubly protonated for HIS (i.e. HIP). Initial protonation states can be specified using the -states flag followed by a comma and/or whitespace-delimited list of initial protonation states (see below for more about protonation state definitions) as follows:

```
cpinutil.py -p prmtop -igb 2 -states 1 3 0 0 0 1 > cpin
```

Note that if a list of states is provided, it must match exactly the number of residues that cpinutil has found to titrate based on the restrictions put on the command line. The -system flag can be used to provide a name for the titrating system. This is purely cosmetic and has no effect on your simulations.

```
cpinutil.py -p prmtop -igb 2 -system HEWL > cpin
```

A number of flags are available for filtering which residues are included in the cpin file. All residues in the cpin file, and only the residues in the cpin file, will be titrated. In general it is safe to exclude TYR and LYS for acidic simulations and GL4 and AS4 for basic simulations. HIP should be included in all except very acidic simulations. Note that there is currently no support for titrating N or C terminal residues. If you have an N or C terminal residue with a titratable sidechain, you should explicitly exclude it from the cpin file. The `-resnum` flag may be used to specify which residue numbers should be retained; all others are deleted. Conversely, the -notresnum flag can be used to specify which residue numbers are deleted; all others are retained. Residue number refers to the numbering in the PDB file, not the index number among titrating residues. Similarly, `-resname` and `-notresname` can be used to filter by residue type. For instance, `-notresname TYR,LYS` would eliminate basic residues from the cpin file. The `-minpKa` and `-maxpKa` flags can be used to filter out residues whose reference pKas do not satisfy that criteria. For example, `-minpKa 5.0` will exclude all AS4 and GL4 residues from titrating.

You can get a full list of all available titratable residues using the `--list` argument to cpinutil.py, and you can get a full description of reference energies and charge vectors for any residue using the `--describe` argument. The full usage statement for cpinutil.py (accessible via `-h/--help`) is shown on the next page.

```
usage: cpinutil.py [Options]
optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program’s version number and exit
  -d, --debug           Enable verbose tracebacks to debug this program
Output files:
  -o FILE, --output FILE
                        Output file. Defaults to standard output
  -op FILE, --output-prmtop FILE
                        For explicit solvent simulations, a custom set of
                        radii are necessary to obtain reasonable results for
                        carboxylate pKas (e.g., AS4 and GL4 residues). If
                        specified, this file will be the prmtop compatible
                        with the reference energies in the printed cpin file.
Required Arguments:
  -p FILE               Topology file to be used in constant pH simulation
Simulation Options:
  -igb IGB              Generalized Born model which you intend to use to
                        evaluate dynamics (or protonation state swaps).
                        Default is 2.
Residue Selection Options:
  -resnames [RES [RES ...]]
                        Residue names to include in CPIN file
  -notresnames [RES [RES ...]]
                        Residue names to exclude from CPIN file
  -resnums [NUM [NUM ...]]
                        Residue numbers to include in CPIN file
  -notresnums [NUM [NUM ...]]
                        Residue numbers to exclude from CPIN file
  -minpKa pKa           Minimum reference pKa to include in CPIN file
  -maxpKa pKa           Maximum reference pKa to include in CPIN file
System Information:
  -states [NUM [NUM ...]]
                        List of default states to assign to titratable
                        residues
  -system <system name>
                        Name of system to titrate. No effect on simulation.
Residue Information:
  If any options here are used, no CPIN file will be written. These
  arguments take precedence and are mutually exclusive with each other.
  --describe [RESNAME [RESNAME ...]]
                        Print out the details of given residues
  -l, --list            List all titratable residues
This program will read a topology file and generate a cpin file for constant
pH simulations with sander
```

22.3. Running at constant pH

22.3.1. Running at constant pH in implicit solvent

Running constant pH simulations in either *sander* or *pmemd* has few differences from normal operation. In the mdin file, you must set *icnstph*=1 to turn on constant pH in implicit solvent. *solvph* is used to set the solvent pH value. You must also specify the period for Monte Carlo steps, *ntcnstph* (the number of steps between protonation state change attempts). Note that only one residue is examined on each step, so you should decrease the step period as the number of titrating residues increases to maintain a constant effective step period for each residue. We have seen good results with fairly short periods, in the neighborhood of 100 fs effective period for each residue (e.g. *ntcnstph*=5, *dt*=0.002 with about 10 residues titrating).

Constant pH MD techniques employ a reference (model) compound to compute relative free energy differences between the various protonation states through a thermodynamic cycle (see Figure 22.1). The free energy of the protonation state change in the model compound that is necessary to yield the correct pK$_a$ is pre-computed for each protonation state change. This so-called *reference energy* is printed to the cpin file by *cpinutil.py*. In order to obtain sensible results, you ***must*** run your simulations with the same potential energy function for your system that was used to derive these reference energies (or alternatively rederive the reference energies with the potential you wish to use).

The reference energies were derived using the following parameters:

```
cut=30.0, igb=#, saltcon=0.1, nrespa=1,
temp0=300.0, ntc=2, ntf=2
```

where # is the value passed to the *cpinutil.py* program. In particular, care should be taken when modifying the *igb*, *saltcon*, *nrespa*, or *temp0* parameters (*nrespa* should never be changed). The cutoff, 30 Å, is effectively infinite for the (very small) model compounds, so using any reasonable cutoff—including an infinite cutoff—is valid. The *ff99SB* force field was used to parametrize the model compounds. Using other force fields should be validated before you run simulations. If the charge scheme is the same as ff99SB (e.g., ff14SB), chances are good that the reference energies will still be valid. Other force fields (e.g., ff03 and ff13) that have different charge definitions require recalculating the reference energies.

The model compounds have the sequence ACE-X-NME, where ACE is a neutral acetyl capping group, X is the titratable residue, and NME is a neutral methylamine capping group. Both ACE and NME are provided in the standard Amber residue libraries.

Some additional command line flags have been added to *sander* to support constant pH operation. The cpin file must be specified using the -cpin option. Additionally, a history of the protonation states sampled is written to the filename specified by -cpout. Finally, a constant pH restart file is written to the filename specified by -cprestrt. This is used to ensure that titrating residues retain the same protonation state when the simulation is restarted. The constant pH restart file is a cpin-format file, and should be used as the cpin file when restarting the simulation. It will generally be longer than the original cpin file, as it contains some amount of zeroed data. The only difference between the cprestrt file created at the end of a simulation and the cpin file used to start it will be the RESSTATES array. Note that due to compiler-dependence of the namelist implementation, cprestrt files may differ from computer to computer.

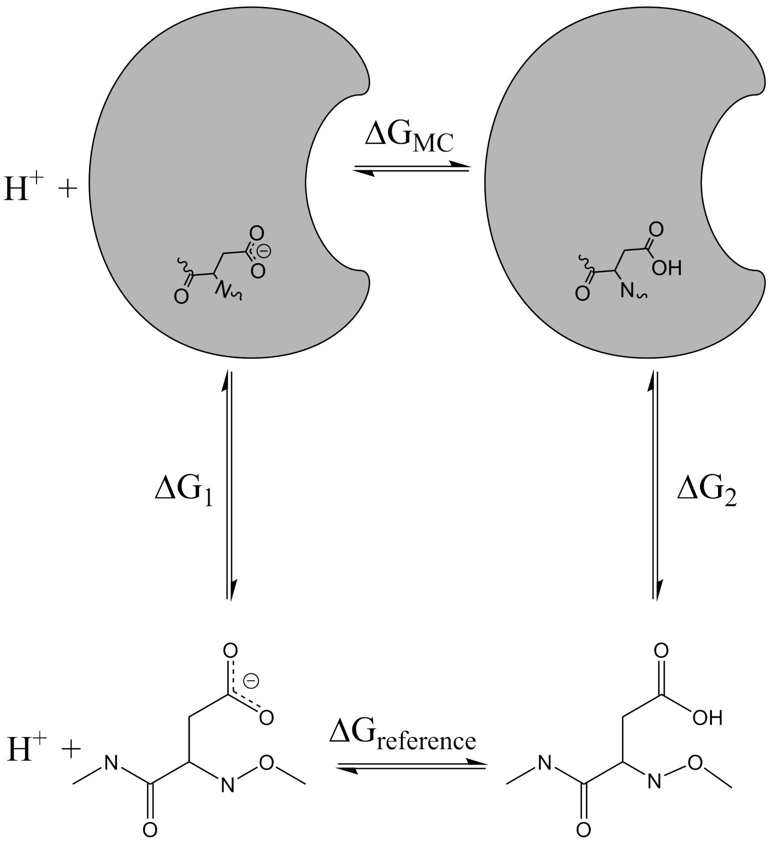


Figure 22.1.: *Thermodynamic cycle used in CpHMD simulations. The energy difference between the two protonation states computed by* sander *is equal to the difference* $\Delta G_1 - \Delta G_2$ *and* $\Delta G_{MC} = \Delta G_2 - \Delta G_1 - \Delta G_{reference}$ *is used to evaluate the Metropolis Monte Carlo criteria for the proposed change in protonation state(s).*

22.3.2. Running at constant pH in explicit solvent

The hybrid molecular dynamics/Monte Carlo technique used in the implicit solvent calculations will not work in explicit solvent because all protonation state changes will be opposed by the solvent orientation around the existing protonation states. To work around this limitation while allowing MD to be propagated in explicit solvent, protonation state changes are still attempted using a Generalized Born implicit solvent model. [484] The workflow, shown in Figure 22.2, involves running MD for `ntcnstph` steps, stripping the solvent and ions, attempting protonation state changes for each titratable residue in random order, and restoring the solvent for running solvent relaxation dynamics if any protonation states have changed before resuming MD.

The modifications needed to run explicit solvent simulations at constant pH are similar to the modifications needed to run implicit solvent simulations at constant pH, with some small differences highlighted here. We found that the existing GB radii defined for carboxylate oxygens is too large for the titratable residues AS4 and GL4. The reason is that the 4 hydrogen atoms in the carboxylate groups are all assigned an intrinsic solvent radius that contributes significantly to the effective radii of the carboxylate oxygens. To compensate, the intrinsic GB radii of AS4 and GL4 carboxylate oxygens must be reduced such that the effective radius is closer to the carboxylate oxygen atoms of an ASP or GLU residue. The *cpinutil.py* script that generates the cpin file has been modified to make the necessary changes to the topology file (which can be written with the new “`-op`” flag that was added for this purpose). An example command-line used to set up a constant pH simulation in explicit solvent for carboxylates is:

```
cpinutil.py -igb 2 -resnames AS4 GL4 -p <tleap_prmtop> \
                  -op <new_radii_prmtop>
```

In the above command, `new_radii_prmtop` is generated and must be used for constant pH simulations. In addition to the modified topology file you need for CpHMD in explicit solvent, there is an additional parameter, *ntrelax*, that defines the number of solvent relaxation steps that will be performed following successful protonation state changes. In general, we've found that while ca. 4 ps is required to generate a truly relaxed solvent distribution, 200 fs is sufficient to account for the bulk of the solvent relaxation.

Another difference with respect to implicit CpHMD simulations is that a protonation state change attempt is carried out for each residue in random order. This is done to allow protonation state change attempts to be done far less frequently to limit the amount of MD time that is consumed by the solvent relaxation dynamics. The following input variables should be used in your *sander* or *pmemd* input file.

```
icnstph=2, ntcnstph=100, ntrelax=200,
solvph=6.4, saltcon=0.1, temp0=300.0,
ntc=2, ntf=2
```

Notice that the value of `icnstph` is 2, which indicates that CpHMD should be run in explicit solvent. The `ntrelax` flag will run solvent relaxation dynamics (in which the non-solvent is held fixed) for 200 steps. The `saltcon` variable controls the salt concentration for the GB calculations. It has no effect on the dynamics, but is required for consistency with the reference energy of the model compound.

22.4. Analyzing constant pH simulations

As the simulation progresses, the protonation states that are sampled are written to the cpout file. A section of a cpout file is included here:

```
Solvent pH: 2.00000
Monte Carlo step size: 2
Time step: 0
Time: 0.000
Residue 0 State: 1
Residue 1 State: 0
Residue 2 State: 1
Residue 3 State: 0
Residue 4 State: 1
Residue 5 State: 0
Residue 2 State: 0
Residue 4 State: 0
Residue 0 State: 3
Residue 1 State: 0
Residue 0 State: 0
```

One record is written on each Monte Carlo step. Each record is terminated by a blank line. There are two types of records, full records and delta records. Full records, like the one shown above, lists the solvent pH, MC step size, current time step, and current time before listing every residue in the system. Full records are written on the first step and every ntwx steps afterwards so as to coincide with the frames written to the trajectory. Delta records list only those residues that were titrated (single or double lines for implicit solvent or a list of every residue for explicit solvent). Note that in some cases, the protonation state for a delta record may be the same as that in an earlier record: this indicates that the Monte Carlo protonation move was rejected for that residue. The residue numbers in cpout are indices over the titrating residues included in the cpin file; cpout files must be analyzed in conjunction with the cpin to map these indices back to the original system.

The Perl script *calcpka.pl* is provided in `$AMBERHOME/AmberTools/src/etc` as an example parser for the cpout format used for calculating predicted pK_a values from cpout files. The *cphstats* program was written to replace calcpka.pl. Its usage is described in Section 22.7.

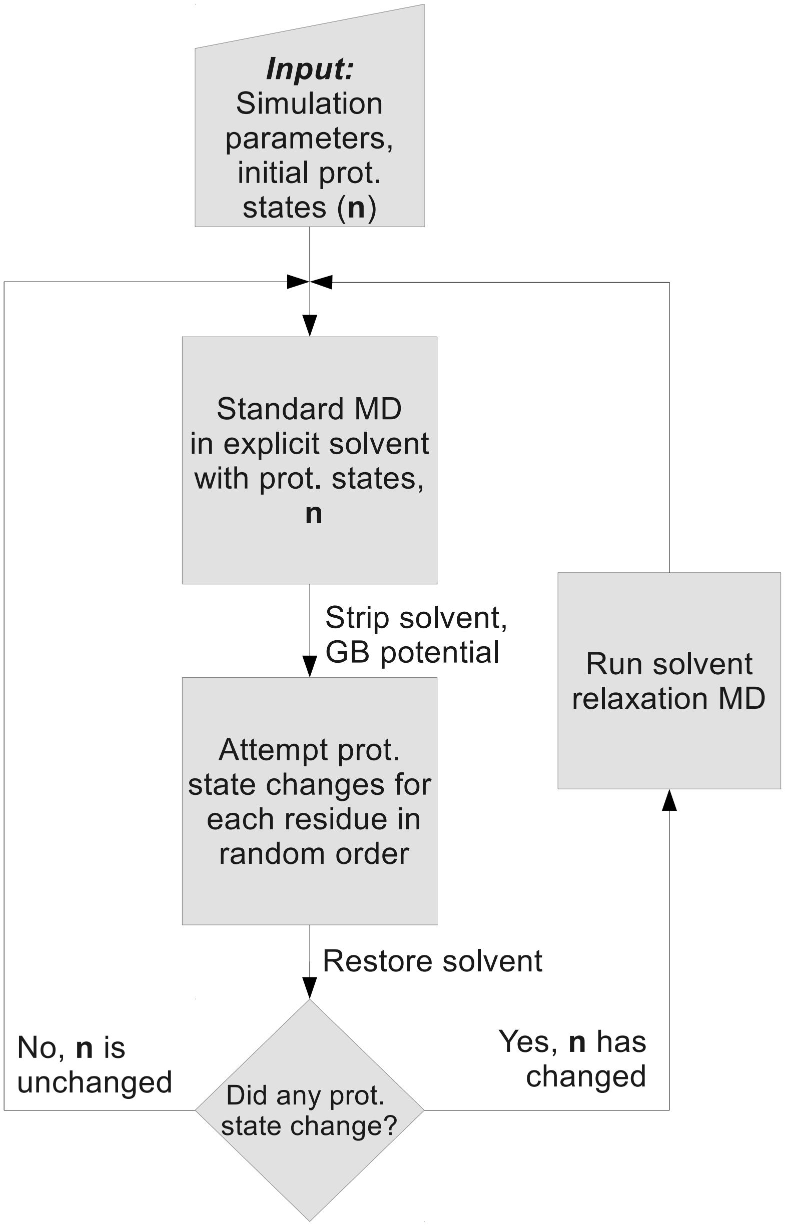

Input:
Simulation parameters, initial prot. states (**n**)
Standard MD in explicit solvent with prot. states, **n**
Strip solvent, GB potential
Attempt prot. state changes for each residue in random order
Restore solvent
Did any prot. state change?
No, **n** is unchanged
Yes, **n** has changed
Run solvent relaxation MD

Figure 22.2.: *Workflow for constant pH MD simulations in explicit solvent.*

22.5. Extending constant pH to additional titratable groups

There are two major components to defining a new titrating group for constant pH. First you must define the partial charges for each atom in the residue for each protonation state. Then you must set the relative energies of each state.

Defining charge sets

Partial charges are most easily calculated using Antechamber and Gaussian. You must set up a model to calculate charges for each protonation state. If the titrating group you are defining is a polymer subunit (e.g. amino acid residue), you must adjust the charges on atoms that have bonded interactions (including 1-4) with atoms in neighboring residues. The charges on these atoms must be changed so they are constant across all protonation states - otherwise relative energies of protonation states become sequence dependent.cmmr an amino acid, this means that all backbone atoms must have constant charges. For the residues defined here, we arbitrarily selected the backbone charges of the protonated state to be used across all protonation states. The total charge difference between states should remain 1; we achieved this by adjusting the charge on the beta carbon.

Calculating relative energies

Relative energies are used to calibrate the method such that when a model compound is titrated at pH equal to its pK_a, the energies (and thus populations) of the protonated and deprotonated states are equal.calcpkamalenergies of the different protonation states are calculated using thermodynamic integration of a model compound between the charge sets defined for the different protonation states. The model compound should be a small molecule that mimics the bonded environment of the titratable group of interest, and for which experimental pK a data are available. For instance, the model compound for an amino acid X is generally ACE-X-NME; the model compound for a ligand might be the free ligand. The thermodynamic integration calculations must be performed using exactly the same parameters and force field as you plan to use in your constant pH simulations. Once the relative energies of the states are calculated by thermodynamic integration, the energy difference must be adjusted to account for the pKa : the energy of the more protonated states should be increased by $pK_aRT\ln(10)$.

For example suppose one were developing a model for an artificial amino acid, ART, with pK a 3.5 and two protonation states: ARP, having one proton and ARD having zero protons. After calculating partial charges as above, you would construct a model compound having the sequence ACE-ARP-NME and generate a prmtop file where the ARP charges were perturbed to the ARD values (this can be done using the ParmEd program bundled with AmberTools). You would then use sander to perform thermodynamic integration between ARP and ARD. Suppose that this showed that the energy of ARD relative to ARP was -6.3 kcal/mol. You would assign a relative energy of -6.3 to ARD and a relative energy of 3.5RTln(10) to ARP.

Testing the titratable group definitions

Prior to large scale use of your new titratable group definition, it's a good idea to test it by performing a constant pH simulation on your model compound, with pH set to the model pK_a. Doing this requires generation of a cpin file, so this is a good time to define your titratable residue to to *cpinutil.py*. These definitions are found in $AMBERHOME/AmberTools/src/parmed/parmed/amber/titratable_residues.py. The residue name must be added to the list `titratable_residues` at the top of the file. Add your residue definition to the bottom of the file, following the examples of the other residues (and make sure to execute the "check" function on that residue at the end as a way of checking your input. It is also a good idea to use *cpinutil.py* with `--describe` to check that the charge vectors match what you meant to input—the output format using `--describe` is much easier to check than the input in *titratable_residues.py*. The definition of CYS is shown below as an example.

```
# Cysteine
refene1 = _ReferenceEnergy(igb2=77.4666763, igb5=76.2588331, igb8=71.5804519)
refene1.solvent_energies(igb2=77.6041407, igb5=76.2827217)
refene1.set_pKa(8.5, deprotonated=False)
```

```
refene2 = _ReferenceEnergy(igb2=0, igb5=0, igb8=0)
refene2.solvent_energies()
CYS = TitratableResidue('CYS', ['N', 'H', 'CA', 'HA', 'CB', 'HB2', 'HB3', 'SG',
                        'HG', 'C', 'O'], pka=8.5)
CYS.add_state(protcnt=1, refene=refene1, # protonated
              charges=[-0.4157, 0.2719, 0.0213, 0.1124, -0.1231, 0.1112, 0.1112,
                       -0.3119, 0.1933, 0.5973, -0.5679])
CYS.add_state(protcnt=0, refene=refene2, # deprotonated
              charges=[-0.4157, 0.2719, 0.0213, 0.1124, -0.3593, 0.1122, 0.1122,
                       -0.8844, 0.0, 0.5973, -0.5679])
CYS.check()
```

Reference energies are those calculated from TI (and adjusted if necessary to reproduce experimental pK_as). The reference energies should be calculated for all GB models you plan to support. For all but one of the states, set the pK_a of the residue so that the reference energy is properly adjusted to the given pK_a. If the state to which you are applying that reference energy is protonated, then set deprotonated=False, as shown above. Otherwise, set deprotonated=True.

You should always titrate your model compound to make sure that calculated pK_as for the model compound match experiment. The TI values are typically sufficient, but some residues may require adjustments.

22.6. Constant pH MD Replica Exchange

Running constant pH replica exchange simulations can be performed in either implicit or explicit solvent. There is no difference in the replica exchange setup between running in implicit or explicit solvent. This method is described in Section 21.5.6 above. We have found that pH-REMD dramatically improves protonation state and conformational state sampling, so we suggest using it whenever possible.

22.7. cphstats

The `cphstats` command-line program was written by Jason Swails to compute protonation state statistics from constant pH simulations (in both implicit and explicit solvent). Written in C++, it is between one and two orders of magnitude faster than the original `calcpka.pl` script written in Perl and roughly twice as fast as the `calcpka` program written in Fortran 90. You can access a list and description of all available command-line flags using the `--help` flag, whose output is shown below.

```
Usage: cphstats [-O] [-V] [-h] [-i <cpin>] [-t] [-o FILE] [-R FILE -r INT]
             [--chunk INT --chunk-out FILE] [--cumulative --cumulative-out FILE]
             [-v INT] [-n INT] [-p|-d] [--calcpka|--no-calcpka] [--fix-remd]
             [--population FILE] [-c CONDITION -c CONDITION -c ...]
             [--conditional-output FILE] [--chunk-conditional FILE]
             cpout1 [cpout2 [cpout3 ...] ]
General Options:
    -h, --help      Print this help and exit.
    -V, --version   Print the version number and exit.
    -O, --overwrite
                    Allow existing outputs to be overwritten.
    --debug         Print out information about the files that are
                    being read in and used for the calculations.
    --expert        I will consider you an expert user and NOT warn
                    you if you try to compute statistics from REMD-based
                    files before using --fix-remd [NOT default behavior]
    --novice        I will warn you if you try to use REMD-based files
                    to compute statistics. [Default behavior]
Input Files and Options:
    -i FILE, --cpin FILE
                  Input cpin file (from sander) with titrating residue
                  information.
    -t FLOAT, --time-step FLOAT
                  This is the time step in ps you used in your simulations.
                  It will be used to print data as a function of time.
                  Default is 2 fs (0.002)
Output Files:
    -o FILE, --calcpka-output FILE
                  File to which the standard `calcpka'-type statistics
                  are written. Default is stdout
    -R FILE, --running-avg-out FILE
                  Output file where the running averages of time series
                  data for each residue is printed (see [Output Options]
                  below for details). Default is [running_avgs.dat]
    --chunk-out FILE
                  Output file where the time series data calculated
                  over chunks of the simulation are printed (see
                  [Output Options] below for details).
                  Default is [chunk.dat]
    --cumulative-out FILE
                  Output file where the cumulative time series data
                  is printed (see [Output Options] below for details).
                  Default is [cumulative.dat]
    --population FILE
                  Output file where protonation state populations are
                  printed for every state of every residue.
    --conditional-output FILE
                  Output file with requested conditional probabilities.
                  Default is [conditional_prob.dat].
    --chunk-conditional FILE
                  Prints a time series of the conditional probabilities over
                  a trajectory split up into chunks.
Output Options:
  These options modify how the output files will appear
    -v INT, --verbose INT
                  Controls how much information is printed to the
                  calcpka-style output file. Options are:
                     (0) Just print fraction protonated. [Default]
                     (1) Print everything calcpka prints.
    -n INT, --interval INT
                  An interval between which to print out time series data
                  like `chunks', `cumulative' data, and running averages.
                  It is also used as the 'window' of the conditional
                  probability time series (--chunk-conditional).
                  Default [1000]
    -p, --protonated
                  Print out protonation fraction instead of deprotonation
                  fraction in time series data (Default behavior).
    -d, --deprotonated
                  Print out deprotonation fraction instead of protonation
                  fraction in time series data.
    -a, --pKa     Print predicted pKas (via Henderson-Hasselbalch) in place
                  of fraction (de)protonated. NOT default behavior.
Analysis Options:
  These options control which analyses are done. By default, only
  the original, calcpka-style analysis is done.
```

```
    --calcpka       Triggers the calcpka-style output [On by default]
    --no-calcpka    Turns off the calcpka-style output
    -r WINDOW, --running-avg WINDOW
                    Defines a window size for a moving, running average
                    time series. <WINDOW> is the number of MD steps (NOT
                    the number of MC exchange attempts).
    --chunk WINDOW
                    Computes the time series data over a chunk of the
                    simulation of size <WINDOW> time steps. See above for
                    details.
    --cumulative    Computes the cumulative average time series data (see above
                    for options) over the course of the trajectory.
    --fix-remd PREFIX
                    This option will trigger cphstats to reassemble the
                    titration data into pH-specific ensembles. This
                    is an exclusive mode of the program---no other
                    analyses will be done.
    -c CONDITIONAL, --conditional CONDITIONAL
                    Evaluates conditional probabilities. CONDITIONAL should be a
                    string of the format:
                          <resid>:<state>,<resid>:<state>,...
                      or
                          <resid>:PROT,<resid>:DEPROT,...
                      or
                          <resid>:<state1>;<state2>,<resid>:PROT,...
                    Where <resid> is the residue number in the prmtop (NOT the
                    cpin) and <state> is either the state number or (p)rotonated
                    or (d)eprotonated, case-insensitive
This program analyzes constant pH output files (cpout) from Amber.
These output files can be compressed using gzip compression. The
compression will be detected automatically by the file name extension.
You must have the gzip headers for this functionality to work.
```

22.7.1. Standard statistics

The standard output of cphstats is the same as that for the `calcpka` and `calcpka.pl` programs that came before. An example from a protein with 10 titratable residues is shown below.

```
Solvent pH is    4.000
GL4 7   : Offset -0.448  Pred  3.552  Frac Prot 0.263 Transitions     91163
HIP 15  : Offset  2.036  Pred  6.036  Frac Prot 0.991 Transitions      6194
AS4 18  : Offset -1.063  Pred  2.937  Frac Prot 0.080 Transitions     41490
GL4 35  : Offset  2.113  Pred  6.113  Frac Prot 0.992 Transitions      5458
AS4 48  : Offset -1.365  Pred  2.635  Frac Prot 0.041 Transitions     17596
AS4 52  : Offset -1.123  Pred  2.877  Frac Prot 0.070 Transitions     25306
AS4 66  : Offset -1.689  Pred  2.311  Frac Prot 0.020 Transitions      7334
AS4 87  : Offset -1.757  Pred  2.243  Frac Prot 0.017 Transitions      8976
AS4 101 : Offset  0.342  Pred  4.342  Frac Prot 0.687 Transitions    105377
AS4 119 : Offset -1.894  Pred  2.106  Frac Prot 0.013 Transitions      6700

Average total molecular protonation:   4.174
```

The external pH that was set in `sander` or `pmemd` is shown at the top, followed by each of the residues with their name and number as they appear in the topology file. The computed pK_a values printed by cphstats are computed by fitting to the Hendersen-Hasselbalch equation (Eq. 22.1). The values printed in the standard output are defined below:

Offset Difference in pK units that the predicted pK_a is from the solvent pH.

Pred The predicted pK_a computed from the fraction protonated and the pH in Eq. 22.1.

Frac Prot The total fraction of the simulation that the residue spent in its 'protonated' form.

Transitions The number of times that the total number of 'active' protons on the titratable residue changed following a protonation state change attempt. This does *not* count when the protonation state changed between two tautomers or protomers with the same number of protons. For instance, the switching from the HID to the HIE tautomers of histidine does not count. Nor does switching from the syn-O1-protonated to the syn-O2-protonated forms of any carboxylate residues.

$$pK_a = pH - \log\left(\frac{1-f_p}{f_p}\right) \tag{22.1}$$

In Eq. 22.1, f_p is the total fraction protonated for a given residue. A more rigorous way of computing the pK_a of a titratable residue is to fit Eq. 22.2—the Hill equation—to the pHs and protonation fractions collected over a full titration curve to compute the best-fit values of the Hill coefficient (n) and computed pK_a. This requires post-processing the output from `cphstats` with your own script or program.

$$f_d = 1 - f_p = \frac{1}{1+10^{n(pK_a - pH)}} \tag{22.2}$$

Example You can analyze as many cpout files as you would like, provided that each cpout file was generated from a simulation run at the same pH as the others. For pH-REMD simulations, a pre-processing stage is initially required, as described in a section 22.7.5. You can direct `cphstats` to print the output to a file with the `-o` flag or have it printed to the screen (stdout) by default. You must provide a `cpin` file with the `-i` flag to calculate any protonation state statistics.

```
cphstats -i cpin pH_4.md1.cpout pH_4.md2.cpout pH_4.md3.cpout \
         -o pH_4.dat
```

22.7.2. Cumulative, running, and "chunk" averages

These options provide a way of monitoring how the ensemble of protonation states evolve during the course of a simulation. Because MD yields insights into the dynamical behavior of molecules, it's often advantageous to monitor the evolution of the protonation state fractions with geometric measurements of the system coordinates. Each option—cumulative, running, and "chunk" averages—can be output as a time series of fraction protonated, fraction deprotonated, or predicted pK_a using the `-p`, `-d`, and `-a` flags, respectively. The details of calculating each of these properties is described in the next sections. The output is printed to a file in the following format:

```
#Time step    GL4 7    HIP 15    AS4 18    GL4 35  Total Avg. Prot.
      1000  0.30693  0.99505  0.00000  0.99505          3.900498
      2000  0.24378  0.99751  0.00000  0.99751          3.962594
      3000  0.23754  0.99834  0.05150  0.99834          4.014975
...
```

The final column is always the total average protonation (note, a protonated histidine counts as '2' protons and a protonated lysine counts as '3', so only differences in total protonation fraction are meaningful). The time step corresponds to the actual MD time step, *not* the interval between protonation state changes.

Cumulative averages

A cumulative average is a time series whose values at time t are calculated according to

$$\langle A \rangle_t = \frac{\int_0^t A(t)dt}{t}$$

such that it represents the average value from time 0 to t. The final average should match the output printed in the standard statistics output of the previous section, which is an average over the entire ensemble. Cumulative averages can be misleading, however, as $\langle A \rangle_t$ changes rapidly when t is small and very slowly as t becomes large. It can give the impression that a property is converging to a particular value when in fact that property is fluctuating a lot.

Example To compute a cumulative average, you must use the `--cumulative` flag to indicate you wish to compute this value. You can control how frequently this quantity is sampled by setting the interval, in MD time steps (*not* protonation state change attempts), using the `-n` flag. The default interval is to print values every 1000 time steps. The longer your simulation is, the less frequently you have to sample points. Data is written to the file cumulative.dat unless a different name is provided with the `--cumulative-out` flag. An example usage is:

```
cphstats -i cpin pH_4.md1.cpout pH_4.md2.cpout pH_4.md3.cpout -n 10000 -p \
         --no-calcpka --cumulative --cumulative-out pH_4_cumulative.dat
```

The `--no-calcpka` flag prevents the standard statistics (previous section) from being computed and printed. The cumulative protonated fraction will be printed to the file pH_4_cumulative.dat with values dumped every 10000 steps.

Running averages

A running average is a time series whose values at time t are calculated according to

$$\langle A \rangle_t = \frac{\int_{t-\sigma}^{t+\sigma} A(t)dt}{2\sigma}$$

such that it represents the average value from $t-\sigma$ to $t+\sigma$—the value 2σ is referred to as the *window* size. The advantage of a running average over a cumulative average is that the shape of the curve at large values of t do not depend on the values near $t=0$. If the interval is smaller than the window size, then adjacent values of $\langle A \rangle_t$ will be comprised of overlapping data points.

Example To compute a running average, you must specify a window size, in MD time steps, with the `-r` flag. The interval—specified with the `-n` flag—controls how frequently samples from the time series are saved to the output file. An example usage is

```
cphstats -i cpin pH_4.md1.cpout pH_4.md2.cpout pH_4.md3.cpout \
         -n 2000 -r 10000 -R pH_4_runningavg.dat -d
```

This command will compute the running average of the deprotonation fraction (because of the `-d` flag) every 2000 time steps with a window size of 10000 time steps—the average will be computed from the 5000 steps before time t to 5000 steps after time t every 2000 steps. It may be important to note that any portion of the window that extends before $t=0$ or after the last time step of the simulation is simply truncated. In this example, the running average data is printed to the file pH_4_runningavg.dat and the standard statistical output is printed to the screen.

"Chunk" averages

A "chunk" average is a time series in which the trajectory is segmented into separate chunks of specified size 2σ time steps. The average value is then calculated according to

$$\langle A \rangle_t = \frac{\int_{t-\sigma}^{t+\sigma} A(t)dt}{2\sigma}$$

Indeed, "chunk" averages are simply a special case of running averages in which the times for which $\langle A \rangle_t$ are computed are the center points of the time chunks. In this analysis, every point of the simulation is uniquely assigned to a single chunk (so no overlap is possible like there is for running averages with a window size larger than twice the interval).

Example Unlike the cumulative and running average analyses, the interval is not used for “chunk” averaging. The chunk size, in MD time steps, simultaneously specifies the size of the simulation to use in the average as well as the positions of the points in the generated time series. An example chunk analysis is

```
cphstats -i cpin pH_4.md1.cpout pH_4.md2.cpout pH_4.md3.cpout --pKa \
         --no-calcpka --chunk 100000 --chunk-out pH_4_chunks.dat
```

This command will break the trajectory into 100,000 time step-chunks and compute the pK_a of each residue for each chunk according to Eq. 22.1. In this example, the “chunk” pK_as are printed to the file pH_4_chunks.dat.

22.7.3. Populations

If you specify a file with the `--populations` flag, the population of every residue in every state computed over the whole trajectory will be printed to the specified file. An example command is shown below

```
cphstats -i cpin pH_4.md1.cpout pH_4.md2.cpout pH_4.md3.cpout \
         --populations populations.dat
```

The populations.dat file will look something like the following:

```
  Residue Number     State  0     State  1     State  2     State  3     State  4
-----------------------------------------------------------------------------------
Residue: GL4 7    0.642697 (0) 0.182143 (1) 0.003043 (1) 0.168653 (1) 0.003465 (1)
Residue: HIP 15   0.982722 (2) 0.017278 (1) 0.000000 (1)
Residue: AS4 18   0.986247 (0) 0.006043 (1) 0.000075 (1) 0.007635 (1) 0.000000 (1)
Residue: GL4 35   0.052398 (0) 0.437664 (1) 0.025815 (1) 0.476629 (1) 0.007495 (1)
Residue: AS4 48   0.995087 (0) 0.001435 (1) 0.000085 (1) 0.003393 (1) 0.000000 (1)
Residue: AS4 52   0.973672 (0) 0.007520 (1) 0.002705 (1) 0.015465 (1) 0.000638 (1)
Residue: AS4 66   0.999612 (0) 0.000165 (1) 0.000000 (1) 0.000223 (1) 0.000000 (1)
Residue: AS4 87   0.948970 (0) 0.023918 (1) 0.001240 (1) 0.025235 (1) 0.000638 (1)
Residue: AS4 101  0.678859 (0) 0.160313 (1) 0.002295 (1) 0.153793 (1) 0.004740 (1)
Residue: AS4 119  0.971682 (0) 0.015043 (1) 0.000623 (1) 0.011888 (1) 0.000765 (1)
```

In the example output, each residue except for histidine 15 is a carboxylate that has 5 available states: deprotonated and protonated on the syn- or anti- positions of either carboxylate oxygen. You can use the `--describe` flag for `cpinutil.py` to get a detailed description of what each state is. The decimal numbers shown are the fraction of the time that the residue spent in the specified protonation state. The integer in parentheses next to the protonation state population is the number of titratable protons that are present in that state. Notice that histidine 15 has only 3 states. The first is the doubly-protonated state whereas the other two states are singly-protonated at the N^{δ} and N^{ε} positions, respectively. This output gives a more detailed view of *where* the protons are during the simulation.

22.7.4. Conditional probabilities

It is frequently the case that an enzyme’s or ribozyme’s catalytic mechanism depends on two titratable residues having specific protonation states to fulfill their role as either a proton donor or acceptor in the mechanism. When one residue is a proton acceptor and the other a proton donor, there is an ‘optimum’ pH at which the fraction of proteins that have the correct set of protonation states will be a maximum and the catalytic rate is lowered when the pH is either raised or reduced from this optimum value, since we typically assume that the only catalytically active state is the one with the catalytically ‘correct’ set of protonation states. By assuming that each residue titrates independently—that is that the protonation state of one does not affect the protonation state of the other—we can derive a pH-rate profile for the enzyme or ribozyme by using the pK_a of each residue to compute fraction of time each residue spends in its catalytically active protonation state and simply multiply those fractions for each residue to arrive at the conditional probability.

If, however, the catalytic residues titrate cooperatively or anticooperatively, the conditional probabilities cannot be computed as a simple product of the individual probabilities. By directly capturing the coupling between dynamics and titration of all titratable residues, CpHMD and pH-REMD are capable of probing this cooperativity. This section describes how to use `cphstats` to compute conditional probabilities directly from the protonation state data.

Conditional Probability Expressions

This section describes the syntax of the conditional probability expressions that you must use for `cphstats`. The format of the expression is a comma-delimited list of residue:state specifications shown below.

```
<residue 1>:<state specification>,<residue 2>:<state specification>
```

You can list as many residues as you want. A snapshot satisfies the conditional probability criteria if each of the specified residues is in the list of allowable states within the state specifications—unspecified residues can be in any state. The residue specifiers are the residue numbers in the topology file. The state specification can be either a semicolon-delimited list of state numbers or the description “protonated” or “deprotonated.” The parser is case-insensitive and you only have to type up to one letter of either word to trigger recognition of “protonated” or “deprotonated.” Example conditional probability expressions are shown below with accompanying descriptions of what conditional probabilities they define. Individual residue:state specifications are indicated by color.

```
"35:P,52:D"
```

This expression is satisfied if residue 35 is protonated at the same time that residue 52 is deprotonated.

```
"35:Prot,52:1;3,15:1"
```

This expression is satisifed if residue 35 is protonated and residue 52 is in either state 1 or 3 and residue 15 is in state 1. Other residues can be in any state.

Examples

You can specify conditional probability expressions following the `-c` flag on the command-line. You can specify as many expressions as you want, as long as each one is preceded by `-c` or `--conditional`. The fraction of states that satisfy each conditional probability is written to the conditional probability output file specified by `--conditional-output` (or conditional_prob.dat by default). An example command-line is shown below

```
cphstats -i cpin pH_4.md1.cpout pH_4.md2.cpout pH_4.md3.cpout \
         -c "35:P,52:D" -c "35:prot,52:1;3,15:1" \
         --conditional-output conditional.dat
```

Example output from conditional.dat is shown below.

```
Conditional Probabilities   Fraction
                35:P,52:D   0.982922
      35:prot,52:1;3,15:1   0.000290
```

For the sake of comparison, the standard statistics are shown below.

```
Solvent pH is    4.000
GL4 7   : Offset -0.167  Pred  3.833  Frac Prot 0.405 Transitions     31378
HIP 15  : Offset  1.391  Pred  5.391  Frac Prot 0.961 Transitions      5578
AS4 18  : Offset -1.851  Pred  2.149  Frac Prot 0.014 Transitions      1910
GL4 35  : Offset  2.432  Pred  6.432  Frac Prot 0.996 Transitions       391
AS4 48  : Offset -2.282  Pred  1.718  Frac Prot 0.005 Transitions       570
AS4 52  : Offset -1.855  Pred  2.145  Frac Prot 0.014 Transitions       438
AS4 66  : Offset -1.998  Pred  2.002  Frac Prot 0.010 Transitions       698
AS4 87  : Offset -1.489  Pred  2.511  Frac Prot 0.031 Transitions      1239
AS4 101 : Offset -0.478  Pred  3.522  Frac Prot 0.250 Transitions     19924
AS4 119 : Offset -1.516  Pred  2.484  Frac Prot 0.030 Transitions      2408

Average total molecular protonation:   3.716
```

In this example, residue 35 is protonated while residue 52 is deprotonated 98.2922% of the time. Assuming that the two residues titrate independently, we would calculate this conditional probability to be $0.996 \times (1 - 0.014) \times 100\% = 98.2056\%$. This indicates that these residues titrate independently at pH 4.

Conditional probability "chunks"

`cphstats` currently supports creating time series of conditional probabilities by breaking the trajectory into equal-sized "chunks" and computing the conditional probability over that chunk (see Sec. 22.7.2 for details). To perform this analysis, specify a conditional "chunk" output file with the `--chunk-conditional` flag. The `-n` flag is used to define the size of the chunk (the same flag used to define the time series interval for cumulative and running averages). The format of the output file is the same as that shown in Sec. 22.7.2 for the other time series, but the columns are labeled with the conditional probability expression instead of the residue name and number. An example usage is shown below.

```
cphstats -i cpin pH_4.md1.cpout pH_4.md2.cpout pH_4.md3.cpout \
         -c "35:P,52:D" -c "35:prot,52:1;3,15:1" -n 100000 \
         --chunk-conditional chunk_conditional.dat
```

This example will break the simulation into 100,000-time-step chunks and compute the two conditional probabilities over each chunk, writing the results to chunk_conditional.dat.

22.7.5. Processing pH-REMD cpout files

Replica exchange in pH-space is performed by attempting to exchange solution pH between replicas. Like with temperature-based REMD, this means that individual replicas do not keep the same pH throughout the entire simulation. Since many of the analyses described in the previous sections pertain to ensembles at one pH, they are not readily applicable to the raw output from pH-REMD simulations and must often be pre-processed before being analyzed. This section describes only how to do the preprocessing. You are expected to be familiar with the content of Subsection 21.5.6 above.

Re-ordering cpout files

You can use `cphstats` to generate pH-specific cpout files from pH-REMD replica cpout files using the `--fix-remd` flag. To do so, you must provide a file name prefix to which the suffix `.pH_X` will be appended as well as the cpout files from every replica for a single simulation (and only a single simulation). If you ran the simulation in multiple steps, using restarts from the previous simulation to start the next, you will have to run the command described here for each segment of the total simulation separately. No cpin file is needed for this step. An example usage is shown below:

```
cphstats --fix-remd cpout pH_1.md1.cpout pH_2.md1.cpout pH_3.md1.cpout \
                          pH_4.md1.cpout pH_5.md1.cpout pH_6.md1.cpout
```

This command will create the files cpout.pH_1.00, cpout.pH_2.00, cpout.pH_3.00, cpout.pH_4.00, cpout.pH_5.00, and cpout.pH_6.00 that can subsequently be used for the analyses described in the previous section.

If you attempt to use REMD cpout files without fixing them in the previous analyses, you will receive an error.

Analyzing replica statistics

There are times when you may want to analyze statistics, such as fraction protonated or deprotonated, from a single replica when looking at things like correlation times, for instance. You can disable the REMD file protection using the `--expert` flag, but note that any computed pK_a is based on Eq. 22.1 with a single pH, so they will be meaningless for REMD-based trajectories.

23. NMR, X-ray, and cryo-EM/ET refinement

We find the *sander* module to be a flexible way of incorporating a variety of restraints into a optimization procedure that includes energy minimization and dynamical simulated annealing. The "standard" sorts of NMR restraints, derived from NOE and J-coupling data, can be entered in a way very similar to that of programs like DISGEO, DIANA or X-PLOR; an aliasing syntax allows for definitions of pseudo-atoms, connections with peak numbers in spectra, and the use of "ambiguous" constraints from incompletely-assigned spectra. More "advanced" features include the use of time-averaged constraints, use of multiple copies (LES) in conjunction with NMR refinement, and direct refinement against NOESY intensities, paramagnetic and diamagnetic chemical shifts, or residual dipolar couplings. In addition, a key strength of the program is its ability to carry out the refinements (usually near the final stages) using an explicit-solvent representation that incorporates force fields and simulation protocols that are known to give pretty accurate results in many cases for unconstrained simulations; this ability should improve predictions in regions of low constraint density and should help reduce the number of places where the force field and the NMR constraints are in "competition" with one another.

Since there is no generally-accepted "recipe" for obtaining solution structures from NMR data, the comments below are intended to provide a guide to some commonly-used procedures. Generally speaking, the programs that need to be run to obtain NMR structures can be divided into three parts:

1. *front-end* modules, which interact with NMR databases that provide information about assignments, chemical shifts, coupling constants, NOESY intensities, and so on. We have tried to make the general format of the input straightforward enough so that it could be interfaced to a variety of programs. At TSRI, we generally use the FELIX and NMRView codes, but the principles should be similar for other ways of keeping track of a database of NMR spectral information. As the flow-chart on the next page indicates, there are only a few files that need to be created for NMR restraints; these are indicated by the solid rectangles. The primary distance and torsion angle files have a fairly simple format that is largely compatible with the DIANA programs; if one wishes to use information from ambiguous or overlapped peaks, there is an additional "MAP" file that makes a translation from peak identifiers to ambiguous (or partial) assignments. Finally, there are some specialized (but still pretty straightforward) file formats for chemical shift or residual dipolar coupling restraints.

   There are a variety of tools, besides the ones described below, that can assist in preparing input for structure refinement in Amber.

   - The SANE (Structure Assisted NOE Evaluation) package, *http://ambermd.org/sane.zip*, is widely used at The Scripps Research Institute.[485]
   - If you use Bruce Johnson's NmrView package, you might also want to look at the TSRI additions to that: *http://garbanzo.scripps.edu/nmrgrp/wisdom/pipe/tips_scripts.html*. In particular, the *xpkTOupl* and *starTOupl* scripts there convert NmrView peak lists into the "7-column" needed for input to makeDIST_RST.
   - Users of the MARDIGRAS programs from UCSF can use the *mardi2amber* program to do conversion to Amber format: *http://picasso.ucsf.edu/mardihome.html*

2. *restrained molecular dynamics,* which is at the heart of the conformational searching procedures. This is the part that *sander* itself handles.

3. *back-end* routines that do things like compare families of structures, generate statistics, simulate spectra, and the like. For many purposes, such as visualization, or the running of procheck-NMR, the "interface" to such programs is just the set of PDB files that contain the family of structures to be analyzed. These general-purpose structure analysis programs are available in many locations and are not discussed here. The

principal *sander*-specific tool is *sviol*, which prepares tables and statistics of energies, restraint violations, and the like.

23.1. Distance, angle and torsional restraints

Distance, angle, and other restraints are read from the DISANG file if *nmropt* > 0. Namelist rst ("&rst") contains the following variables; it is read repeatedly until a namelist &rst statement is found with IAT(1)=0, or until reaching the end of the DISANG file.

[In many cases, the user will nkot prepare this section of the input by hand, but will use the auxiliary programs *makeDIST_RST, makeANG_RST* and *makeCHIR_RST* to prepare input from simpler files. See also the programs *cyanarest_to_amberRST* and *nef_to_RST* if you have restraints in Cyana or NEF (NMR Exchange Format) formats.]

23.1.1. Variables in the &rst namelist:

iat(1)→iat(8)

- *If IRESID = 0 (normal operation):* The atoms defining the restraint. Type of restraint is determined (in order) by:
  1. If IAT(3) = 0, this is a distance restraint.
  2. If IAT(4) = 0, this is an angle restraint.
  3. If IAT(5) = 0, this is a torsional (or J-coupling, if desired) restraint or a genereralized distance restraint of 4 atoms, a type of restraint new as of Amber 10 (*sander* only, see below).
  4. If IAT(6) = 0, this is a plane-point angle restraint, a second restraint new as of Amber 10 (*sander* only). The angle is measured between the normal of a plane defined by IAT(1)..IAT(4) and the vector from the center of mass of atoms IAT(1)..IAT(4) to the position of IAT(5). The normal is defined by (r1 - r2) × (r3 - r4), where rn is the position of IAT(n).
  5. If IAT(7) = 0, this is a generalized distance restraint of 6 atoms (see below).
  6. Otherwise, if IAT(1)..IAT(8) are all nonzero, this is a plane- plane angle restraint, a third new restraint type as of Amber 10 (*sander* only, or a generalized distance restraint of 8 atoms (see below). For the plane-plane restraint, the angle is measured between the two normals of the two planes, which are defined by (r1 - r2) × (r3 - r4) and (r5 - r6) × (r7 - r8). In the case of either planar restraint, the plane may be defined using three atoms instead of four simply by using one atom twice.

  If any of IAT(n) are < 0, then a corresponding group of atoms is defined below, and the coordinate- averaged position of this group will be used in place of atom IAT(n). A new feature as of Amber 10, atom groups may be used not only in distance restraints, but also in angle, torsion, the new plane restraints, or the new generalized restraints. 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(8) point to the *residues* containing the atoms comprising the internal. Residue numbers are the absolute in the entire system. In this case, the variables ATNAM(1)..ATNAM(8) must be specified and give the character names of the atoms within the respective residues. If any of IAT(n) are less than zero, then group input will still be read in place of the corresponding atom, as described below.

- *Defaults for IAT(1)→IAT(8) are 0.*

rstwt(1)→rstwt(4) New as of Amber 10 (*sander* only), users may now define a single restraint that is a function of multiple distance restraints, called a "generalized distance coordinate" restraint. The energy of such a restraint has the following form:

$$U = k(w_1|\mathbf{r}_1 - \mathbf{r}_2| + w_2|\mathbf{r}_3 - \mathbf{r}_4| + w_3|\mathbf{r}_5 - \mathbf{r}_6| + w_4|\mathbf{r}_7 - \mathbf{r}_8| - r_0)^2$$

where the weights $w_n$are given in rstwt(1)..rstwt(4) and the positions $\mathbf{r}_n$are the positions of the atoms in iat(1)..iat(8).

Generalized distance coordinate restraints must be defined with either 4, 6, or 8 atoms and 2, 3, or 4 corresponding nonzero weights in rstwt(1)..rstwt(4). Weights may be any positive or negative real number.

If all the weights in rstwt(1)..rstwt(4) are zero and four atoms are given in iat(1)..iat(4) for the restraint, the restraint is a torsional or J-coupling restraint. If eight atoms are given in iat(1)..iat(8) and all weights are zero, the restraint is a plane-plane angle restraint. However, if the weights are nonzero, the restraint will be a generalized distance coordinate restraint.

Default for rstwt(1)..rstwt(4) is 0.0

restraint

New as of Amber 10 (*sander* only), users may now use a "natural language" system to define restraints by using the RESTRAINT character variable. Valid restraints defined in this manner will begin with a "distance( )" "angle( )" "torsion( )" or "coordinate( )" keyword. Within the parentheses, the atoms that make up the restraint are specified. Atoms may be defined either with an explicit atom number or by using ambmask format, namely :(residue#)@(atom name). Atoms may be separated by commas, spaces, or parentheses. Additionally, negative integers may be used if atom groupings are defined in other variables in the namelist as described below. In addition to the principle distance, angle, torsion, and coordinate keywords, Some keywords may be used within the principle keywords to define more complicated restraints. The keyword "plane( )" may be used once or twice within the parentheses of the "angle( ) keyword to define a planar restraint. Defining one plane grouping plus one other atom in this manner will create a plane-point angle restraint as described above. Defining two plane groupings will create a plane-plane angle restraint. The keyword "plane( )" may only be used inside of "angle( )," and is necessary to define either a plane-point or plane-plane restraint.

Within the "coordinate( )" keyword, the user must use 2 to 4 "distance( )" keywords to define a generalized distance coordinate restraint. The "distance( )" keyword functions just like it does when used to define a traditional distance restraint. The user may specify any two atom numbers, masks, or negative numbers corresponding to atom groups defined outside of RESTRAINT. Additionally, following each "distance( )" keyword inside "coordinate( )" the user must specify a real-number weight to be applied to each distance making up the generalized coordinate.

The "com( )" keyword may be used within any other keyword to define a center of mass grouping of atoms. Within the parenthesis, the user will enter a list of atom numbers or masks. Negative numbers, which correspond to externally-defined groups, may not be used.

Any type of parenthetical character, i.e., ( ), [ ], or { }, may be used wherever parentheses have been used above.

The following are all examples of valid restraint definitions:

```
restraint = "distance( (45) (49) )"
= "angle (:21@C5′ :21@C4′ 108)"
= "torsion[−1,−1,−1, com(67, 68, 69)]"
= "angle( −1, plane(81, 85, 87, 90) )"
= "angle(plane(com(9,10),:5@CA,31,32),plane(14,15,15,16))"
= "coordinate(distance(:5@C3′,:6@O5′),−1.0,distance(134,−1),1.0)"
```

There is a 256 character limit on RESTRAINT, so if a particularly large atom grouping is desired, it is necessary to specify a negative number instead of "com( )" and define the group as described below. RESTRAINT will only be parsed if IAT(1) = 0, otherwise the information in IAT(1) .. IAT(8) will define the restraint. *Default for restraint is ' '.*

atnam

If IRESID = 1, then the character names of the atoms defining the internal are contained in ATNAM(1)→ATNAM(8). Residue IAT(1) is searched for atom name ATNAM(1); residue IAT(2) is searched for atom name ATNAM(2); etc. *Defaults for ATNAM(1)→ATNAM(8) are ' '.*

iresid Indicates whether IAT(I) points to an atom # or a residue #. See descriptions of IAT() and ATNAM() above. If RESTRAINT is used to define the internal instead of IAT(), IRESID has no effect on how RESTRAINT is parsed. However, it will affect the behavior of atom group definitions as described below if negative numbers are specified within RESTRAINT. *Default = 0.*

nstep1, nstep2 This restraint is applied for steps/iterations NSTEP1 through NSTEP2. If NSTEP2 = 0, the restraint will be applied from NSTEP1 through the end of the run. Note that the first step/iteration is considered step zero (0). *Defaults for NSTEP1, NSTEP2 are both 0.*

irstyp Normally, the restraint target values defined below (R1→R4) are used directly. If IRSTYP = 1, the values given for R1→R4 define relative displacements from the current value (value determined from the starting coordinates) of the restrained internal. For example, if IRSTYP=1, the current value of a restrained distance is 1.25, and R1 (below) is -0.20, then a value of R1=1.05 will be used. *Default is IRSTYP=0.*

ialtd Determines what happens when a distance restraint gets very large. If IALTD=1, then the potential "flattens out", and there is no force for large violations; this allows for errors in constraint lists, but might tend to ignore constraints that *should* be included to pull a bad initial structure towards a more correct one. When IALTD=0 the penalty energy continues to rise for large violations. See below for the detailed functional forms that are used for distance restraints. Set IALTD=0 to recover the behavior of earlier versions of *sander*. Default value is 0, or the last value that was explicitly set in a previous restraint. This value is set to 1 if *makeDIST_RST* is called with the *-altdis* flag.

ifvari If IFVARI > 0, then the force constants/positions of the restraint will vary with step number. Otherwise, they are constant throughout the run. If IFVARI >0, then the values R1A→R4A, RK2A, and RK3A must be specified (see below). *Default is IFVARI=0.*

ninc If IFVARI > and NINC > 0, then the change in the target values of of R1→R4 and K2,K3 is applied as a step function, with NINC steps/ iterations between each change in the target values. If NINC = 0, the change is effected continuously (at every step). *Default for NINC is the value assigned to NINC in the most recent namelist where NINC was specified. If NINC has not been specified in any namelist, it defaults to 0.*

imult If IMULT=0, and the values of force constants RK2 and RK3 are changing with step number, then the changes in the force constants will be linearly interpolated from rk2→rk2a and rk3→rk3a as the step number changes. If IMULT=1 and the force constants are changing with step number, then the changes in the force constants will be effected by a series of multiplicative scalings, using a single factor, R, for all scalings. *i.e.*

```
rk2a = R**INCREMENTS * rk2
rk3a = R**INCREMENTS * rk3.
```

INCREMENTS is the number of times the target value changes, which is determined by NSTEP1, NSTEP2, and NINC. *Default for IMULT is the value assigned to IMULT in the most recent namelist where IMULT was specified. If IMULT has not been specified in any namelist, it defaults to 0.*

r1→r4, rk2, rk3, r1a→r4a, rk2a, rk3a If IALTD=0, the restraint is a well with a square bottom with parabolic sides out to a defined distance, and then linear sides beyond that. 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 restraint energy $k_2(R-r_2)^2$.
- r2 <= R < r3 E = 0.
- r3 <= R < r4 Parabolic, with restraint energy $k_3(R-r_3)^2$.
- 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-Å2 Force constants for angles are in kcal/mol-rad 2 . (Note that angle positions are specified in degrees, but force constants are in radians, consistent with typical reporting procedures in the literature).

If IALTD=1, distance restraints are interpreted in a slightly different fashion. Again, If R is the value of the restraint in question:

- R < r2 Parabolic, with restraint energy $k_2(R-r_2)^2$.
- r2 <= R < r3 E = 0.
- r3 <= R < r4 Parabolic, with restraint energy $k_3(R-r_3)^2$.
- r4 <= R Hyperbolic, with energy $k_3[b/(R-r_3)+a]$, where $a = 3(r_4-r_3)^2$ and $b = -2(r_4-r_3)^3$. This function matches smoothly to the parabola at $R = r_4$, and tends to an asymptote of ak_3at large R. The functional form is adapted from that suggested by Michael Nilges, *Prot. Eng.* **2,** 27-38 (1988). Note that if *ialtd=1*, the value of r1 is ignored.

ifvari **= 0** The values of r1→r4, rk2, and rk3 will remain constant throughout the run.

> 0 The values r1a, r2a, r3a, r4a, r2ka and r3ka are also used. These variables are defined as for r1→r4 and rk2, rk3, but correspond to the values appropriate for NSTEP = NSTEP2: e.g., if IVARI >0, then the value of r1 will vary between NSTEP1 and NSTEP2, so that, e.g. r1(NSTEP1) = r1 and r1(NSTEP2) = r1a. Note that you *must* specify an explicit value for *nstep1* and *nstep2* if you use this option. *Defaults for r1→r4,rk2,rk3,r1a→r4a,rk2a and rk3a are the values assigned to them in the most recent namelist where they were specified. They should always be specified in the first* &rst *namelist.*

r0, k0, r0a, k0a New as of Amber 10 (*sander* only), the user may more easily specify a large parabolic well if desired by using R0 and K0, and then R0A and K0A if IFVARI > 0. The parabolic well will have its zero at R = R0 and a force constant of K0. These variables simply map the disired parabolic well into r1→r4, rk2, rk3, r1a→r4a, rk2a, and rk3a in the following manner:

- R1 = 0 for distance, angle, and planar restraints, R1 = R0 - 180 for torsion restraints
- R1A = 0 for distance, angle, and planar restraints, R1A = R0A - 180 for torsion restraints
- R2 = R0; R3 = R0
- R2A = R0A; R3A = R0A
- R4 = R0 + 500 for distance restraints, R4 = 180 for angle and planar restraints, R4 = R0 + 180 for torsion restraints
- RK2 = K0; RK3 = K0
- RK2A = K0A; RK3A = K0A

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 nonzero 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.*

igr1(i),i=1→200, igr2(i),i=1→200, ... igr8(i),i..1=1→200 If IAT(n) < 0, then IGRn() gives the atoms defining the group whose coordinate averaged position is used to define "atom n" in a restraint. Alternatively, if RESTRAINT is used to define the internal, then if the nth atom specified is a number less than zero, IGRn() gives the atoms defining the group whose coordinate averaged position is used to define "atom n" in a restraint. If IRESID = 0, absolute atom numbers are specified by the elements of IGRn(). If IRESID = 1, then IGRn(I) specifies the number of the residue containing atom I, and the name of atom I must be specified using GRNAMn(I). A maximum of 200 atoms (1024 if using pmemd) are allowed in any group. Only specify those atoms that are needed. Default value for any unspecified element of IGRn(i) is 0.

fxyz If iat(3)=0 and igr1 and/or igr2 is defined then it is possible to weight the x, y, z components of the force in the restraint to 0 (no force) or 1 (full restraint force). Ex: fxyz=0, 0, 1. This sets no additional restraint force on the x component or y-component of the restraint force, and full z-component restraint force. Default fxyz=1,1,1. Note: When setting fxyz, the r1, r2, r3, r4 values should be set relative to a weighted distance $\sqrt{(w_x * d_x)^2 + (w_y * d_y)^2 + (w_z * d_z)^2}$, so if fxyz=0,0,1 then the only distance taken into account when comparing to r1,r2,r3,r4 is the z distance between the molecule and the center of mass. Note that the DUMPAVE value when outxyz=0 is also just the weighted distance.

outxyz If iat(3)=0 and igr1 and/or igr2 is defined then it is possible to output the x, y, z components of the force in the restraint if outxyz is set to 1. Default outyz=0. When outxyz is set to 1, the components of the distance and total distance are outputed in DUMPAVE in the order of the x-component, y-component, z-component, total distance.

grnam1(i),i=1→200, grnam2(i),i=1→200, ... grnam8(i),i=1→200 If group input is being specified (IGRn(1) $>$ 0), *and* IRESID = 1, then the character names of the atoms defining the group are contained in GRNAMn(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 GRNAMn(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 $\langle r^{-6} \rangle^{-1/6}$ average of all interaction distances to atoms of the group will be used. *Default for IR6 is the value assigned to IR6 in the most recent namelist where IR6 was specified. If IR6 has not been specified in any namelist, it defaults to 0.*

ifntyp If time-averaged restraints have been requested (see DISAVE/ANGAVE/TORAVE above), they are, by default, applied to all restraints of the class specified. Time-averaging can be overridden for specific internals of that class by setting IFNTYP for that internal to 1. IFNTYP has no effect if time-averaged restraint are not being used. *Default value is IFNTYP=0.*

ixpk, nxpk These are user-defined integers than can be set for each constraint. They are typically the "peak number" and "spectrum number" associated with the cross-peak that led to this particular distance restraint. Nothing is ever done with them except to print them out in the "violation summaries", so that NMR people can more easily go from a constraint violation to the corresponding peak in their spectral database. Default values are zero.

iconstr If *iconstr* > 0, (default is 0) a Lagrangian multiplier is also applied to the two-center internal coordinate defined by IAT(1) and IAT(2). The effect of this Lagrangian multiplier is to maintain the initial orientation of the internal coordinate. The rotation of the vector IAT(1)->IAT(2) is prohibited, though translation is allowed. For each defined two-center internal coordinate, a separate Lagrangian multiplier is used. Therefore, although one can use as many multipliers as needed, defining centers should

NOT appear in more than one multiplier. This option is compatible with mass centers (i.e., negative IAT(1) or IAT(2)). ICONSTR can be used together with harmonic restraints. RK2 and RK3 should be set to 0.0 if the two-center internal coordinate is a simple Lagrangian multiplier. An example has been included in $AMBERHOME/example/lagmul.

Namelist &rst is read for each restraint. Restraint input ends when a namelist statement with iat(1) = 0 (or iat(1) not specified) is found. Note that comments can precede or follow any namelist statement, allowing comments and restraint definitions to be freely mixed.

23.2. 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; see the "NMR variables" description in that section.

In many cases, the user will not prepare this section of the input by hand, but will use the auxiliary program *makeDIST_RST* to prepare input from simpler files.

Variables in the &noeexp namelist:

For each submolecule, the namelist "&noeexp" is read (either from *stdin* or from the NOESY redirection file) which contains the following variables. There are no effective defaults for *npeak, emix, ihp, jhp,* and *aexp*: you must specify these.

npeak*(imix)* Number of peaks for each of the "imix" mixing times; if the last mixing time is *mxmix*, set NPEAK(*mxmix+1*) = -1. End the input when NPEAK(1) $<$ 0.

emix*(imix)* Mixing times (in seconds) for each mixing time.

ihp*(imix,ipeak)*, jhp*(imix,ipeak)* Atom numbers for the atoms involved in cross-peak "ipeak" at mixing time "imix"

aexp(imix,ipeak) Experimental target integrated intensity for this cross peak. If AEXP is negative, this cross peak is part of a set of overlapped peaks. The computed intensity is added to the peak that follows; the next time a peak with AEXP $>$ 0 is encountered, the running sum for the calculated peaks will be compared to the value of AEXP for that last peak in the list. In other words, a set of overlapped peaks is represented by one or more peaks with AEXP $<$ 0 followed by a peak with AEXP $>$ 0. The computed total intensity for these peaks will be compared to the value of AEXP for the final peak.

arange*(imix,ipeak)* "Uncertainty" range for this peak: if the calculated value is within ±ARANGE of AEXP, then no penalty will be assessed. Default uncertainties are all zero.

awt*(imix,ipeak)* Relative weight for this cross peak. Note that this will be multiplied by the overall weight given by the NOESY weight change cards in the weight changes section (Section 1). Default values are 1.0, unless INVWT1,INVWT2 are set (see below), in which case the input values of AWT are ignored.

invwt1,invwt2 Lower and upper bounds on the weights for the peaks respectively, such that the relative weight for each peak is 1/intensity if 1/intensity lies between the lower and upper bounds. This is the intensity after being scaled by *oscale*. The inverse weighing scheme adopted by this option prevents placing too much influence on the strong peaks at the expense of weaker peaks and was previously invoked using the compilation flag "INVWGT". Default values are INVWT1=INVWT2=1.0, placing equal weights on all peaks.

omega	Spectrometer frequency, in Mhz. Default is 500. It is possible for different sub-molecules to have different frequencies, but omega will only change when it is explicitly re-set. Hence, if all of your data is at 600 Mhz, you need only set *omega* to 600. in the first submolecule.
taurot	Rotational tumbling time of the molecule, in nsec. Default is 1.0 nsec. Like *omega*, this value is "sticky", so that a value set in one submolecule will remain until it is explicitly reset.
taumet	Correlation time for methyl jump motion, in ns. This is only used in computing the intra-methyl contribution to the rate matrix. The ideas of Woessner are used, specifically as recommended by Kalk & Berendsen.[486] Default is 0.0001 ns, which is effectively the fast motion limit. The default is consistent with the way the rest of the rate matrix elements are determined (also in the fast motion limit,) but probably is not the best value to use, since methyl groups appear to have T1 values that are systematically shorter than other protons, and this is likely to arise from the fact that the methyl correlation time can be near to the inverse of the spectrometer frequency. A value of 0.02 - 0.05 ns is probably better than 0.0001, but this is still an active research area, and you are on your own here, and should consult the literature for further discussion.[487] As with *omega*, *taumet* can be different for different sub-molecules, but will only change when it is explicitly re-set.
id2o	Flag for determining if exchangeable protons are to be included in the spin-diffusion calculation. If ID2O=0 (default) then all protons are included. If ID2O=1, then all protons bonded to nitrogen or oxygen are assumed to not be present for the purposes of computing the relaxation matrix. No other options exist at present, but they could easily be added to the subroutine *indexn*. Alternatively, you can manually rename hydrogens in the *prmtop* file so that they do not begin with "H": such protons will not be included in the relaxation matrix. (*Note:* for technical reasons, the HOH proton of tyrosine must always be present, so setting ID2O=1 will not remove it; we hope that this limitation will be of minor importance to most users.) The *id2o* variable retains its value across namelist reads, *i.e.* its value will only change if it is explicitly reset.
oscale	overall scaling factor between experimental and computed volume units. The experimental intensities are multiplied by *oscale* before being compared to calculated intensities. This means that the weights WNOESY and AWT always refer to "theoretical" intensity scales rather than to the (arbitrary) experimental units. The *oscale* variable retains its value across namelist reads, *i.e.* its value will only change if it is explicitly reset. The initial (default) value is 1.0.

The atom numbers *ihp* and *jhp* are the absolute atom numbers. For methyl groups, use the number of the last proton of the group; for the delta and epsilon protons of aromatic rings, use the delta-2 or epsilon-2 atom numbers. Since this input requires you to know the absolute atom numbers assigned by Amber to each of the protons, you may wish to use the separate *makeDIST_RST* program which provides a facility for more turning human-readable input into the required file for *sander*.

Following the &noeexp namelist, give the Amber "group" cards that identify this submolecule. This combination of "&noeexp" and "group" cards can be repeated as often as needed for many submolecules, subject to the limits described in the *nmr.h* file. As mentioned above, this input section ends when NPEAK(1) $<$ 0, or when and end-of-file is reached.

23.3. Chemical shift restraints

After reading NOESY restraints above (if any), read the chemical shift restraints in namelist *&shf*, or the pseudocontact restraints in namelist *&pcshift*. Reading this input is triggered by the presence of a SHIFTS line in the I/O redirection section. In many cases, the user will not prepare this section of the input by hand, but will use the auxiliary programs *shifts* 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;[488] more recent empirical parametrizations seem to give improved results.[489, 490]
nprot	Number of protons for which penalty functions are to be set up.
iprot(i)	Absolute atom number of the *i-th* proton whose shifts are to be evaluated. For equivalent protons, such as methyl groups or rapidly flipping phenylalanine rings, enter all two or three atom numbers in sequence; averaging will be controlled by the *wt* parameter, described below.
obs(i)	Observed secondary shift for the *i-th* proton. This is typically calculated as the observed value minus a random coil reference value.
shrang(i)	"Uncertainty" range for the observed shift: if the calculated shift is within ±SHRANG of the observed shift, then no penalty will be imposed. The default value is zero for all shifts.
wt(i)	Weight to be assigned to this penalty function. Note that this value will be multiplied by the overall weight (if any) given by the SHIFTS command in the assignment of weights (above). Default values are 1.0. For sets of equivalent protons, give a negative weight for all but the last proton in the group; the last proton gets a normal, positive value. The average computed shift of the group will be compared to *obs* entered for the last proton.
shcut	Values of calculated shifts will be printed only if the absolute error between calculated and observed shifts is greater than this value. *Default = 0.3 ppm.*
nter	Residue number of the N-terminus, for protein shift calculations; *default = 1.*
cter	Residue number of the C-terminus, for protein shift calculations. Believe it or not, the current code cannot figure this out for itself.

In typical usage, the *shifts* program (`http://casegroup.rutgers.edu/shifts.html`) would be used to create this file, with a typical command line:

```
shifts -readobs -sander '::H*' gcg10
```

Sample input and output files are in `$AMBERHOME/test/rdc`.

23.4. Pseudocontact shift restraints

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,

$\Delta\chi_{ax}$ and $\Delta\chi_{rh}$ are the axial and the equatorial (rhombic) anisotropies of the magnetic susceptibility tensor of the j-th paramagnetic center. For a discussion, see Ref. [491].

The PCSHIFT module to be used needs a namelist file which includes information on the magnetic susceptibility tensor and on the paramagnetic center, and a line of information for each nucleus. This module allows to include more than one paramagnetic center in the calculations. To include pseudocontact shifts as constraints in energy minimization and molecular dynamics calculations the NMROPT flag should be set to 2, and a *PCSHIFT=filename* statement entered in the I/O redirection section.

To perform molecular dynamics calculations it is necessary to eliminate the rotational and translational degree of freedom about the center of mass (this because during molecular dynamics calculations the relative orientation between the external reference coordinate system and the magnetic anisotropy tensor coordinate system has to be fixed).This option can be obtained with the NSCM flag of *sander*.

Variables in the pcshift namelist

nprot	number of pseudocontact shift constraints.
nme	number of paramagnetic centers.
nmpmc	name of the paramagnetic atom
optphi(n), opttet(n), optomg(n), opta1(n), opta2(n)	the five parameters of the magnetic anisotropy tensor for each paramagnetic center.
optkon	force constant for the pseudocontact shift constraints

Following this, there is a line for each nucleus for which the pseudocontact shift information is given has to be added. Each line contains :

iprot(i)	atom number of the i-th proton whose shift is to be used as constraint.
obs(i)	observed pseudocontact shift value, in ppm
wt(i)	relative weight
tolpro(i)	relative tolerance ix mltpro
mltpro(i)	multiplicity of the NMR signal (for example the protons of a methyl group have mltprot(i)=3)

Example

Here is a &pcshf namelist example: a molecule with three paramagnetic centers and 205 pseudocontact shift constraints.

```
&pcshf
nprot=205,
nme=3,
nmpmc='FE ',
optphi(1)=-0.315416,
opttet(1)=0.407499,
optomg(1)=0.0251676,
opta1(1)=-71.233,
opta2(1)=1214.511,
optphi(2)=0.567127,
opttet(2)=-0.750526,
optomg(2)=0.355576,
opta1(2)=-60.390,
opta2(2)=377.459,
optphi(3)=0.451203,
opttet(3)=-0.0113097,
optomg(3)=0.334824,
opta1(3)=-8.657,
opta2(3)=704.786,
optkon=30,
iprot(1)=26, obs(1)=1.140, wt(1)=1.000, tolpro(1)=1.00, mltpro(1)=1,
iprot(2)=28, obs(2)=2.740, wt(2)=1.000, tolpro(2)=.500, mltpro(2)=1,
iprot(3)=30, obs(3)=1.170, wt(3)=1.000, tolpro(3)=.500, mltpro(3)=1,
iprot(4)=32, obs(4)=1.060, wt(4)=1.000, tolpro(4)=.500, mltpro(4)=3,
iprot(5)=33, obs(5)=1.060, wt(5)=1.000, tolpro(5)=.500, mltpro(5)=3,
iprot(6)=34, obs(6)=1.060, wt(6)=1.000, tolpro(6)=.500, mltpro(6)=3,
...
...
iprot(205)=1215, obs(205)=.730, wt(205)=1.000, tolpro(205)=.500,
mltpro(205)=1,
/
```

An *mdin* file that might go along with this, to perform a maximum of 5000 minimization cycles, starting with 500 cycles of steepest descent. PCSHIFT=./pcs.in redirects the input from the namelist "pcs.in" which contains the pseudocontact shift information.

```
Example of minimization including pseudocontact shift constraints
&cntrl
ibelly=0,imin=1,ntpr=100,
ntr=0,maxcyc=500,
ncyc=50,ntmin=1,dx0=0.0001,
drms=.1,cut=10.,
nmropt=2,pencut=0.1, ipnlty=2,
/
&wt type='REST', istep1=0,istep2=1,value1=0.,
value2=1.0, /
&wt type='END' /
DISANG=./noe.in
PCSHIFT=./pcs.in
LISTOUT=POUT
```

23.5. Direct dipolar coupling restraints

Energy restraints based on direct dipolar coupling constants are entered in this section. All variables are in the namelist &align; reading of this section is triggered by the presence of a DIPOLE line in the I/O redirection section.

When dipolar coupling restraints are turned on, the five unique elements of the alignment tensor are treated as additional variables, and are optimized along with the structural parameters. Their effective masses are determined by the *scalm* parameter entered in the &cntrl namelist. Unlike some other programs, the variables used are the Cartesian components of the alignment tensor in the axis system defined by the molecule itself: *e.g.* $S_{mn} \equiv \langle (3\cos\theta_m \cos\theta_n - \delta_{mn})/2 \rangle$, where $m,n = x,y,z$, and θ_x is the angle between the x axis and the spectrometer field.[492] The factor of 10^5 is just to make the values commensurate with atomic coordinates, since both the coordinates and the alignment tensor values will be updated during the refinement. The calculated dipolar splitting is then

$$D_{calc} = -\left(\frac{10^{-5}\gamma_i\gamma_j h}{2\pi^2 r_{ij}^3}\right) \sum_{m,n=xyz} \cos\phi_m \cdot S_{mn} \cdot \cos\phi_n$$

where ϕ_x is the angle between the internuclear vector and the x axis. Geometrically, the splitting is proportional to the transformation of the alignment tensor onto the internuclear axis. This is just Eqs. (5) and (13) of the above reference, with any internal motion corrections (which might be a part of S_{system}) set to unity. If there is an internal motion correction which is the same for all observations, this can be assimilated into the alignment tensor. The current code does not allow for variable corrections for internal motion. See Ref. [493] for a fuller discussion of these issues.

At the end of the calculation, the alignment tensor is diagonalized to obtain information about its principal components. This allows the alignment tensor to be written in terms of the "axial" and "rhombic" components that are often used to describe alignment.

Variables in the &align namelist.

ndip Number of observed dipolar couplings to be used as restraints.

id,jd Atom numbers of the two atoms involved in the dipolar coupling.

dobsl, dobsu Limiting values for the observed dipolar splitting, in Hz. If the calculated coupling is less than *dobsl*, the energy penalty is proportional to $(D_{calc} - D_{obs,l})^2$; if it is larger than *dobsu*, the penalty is proportional to $(D_{calc} - D_{obs,lu})^2$. Calculated values between *dobsl* and *dobsu* are not penalized. Note that *dobsl* must be less than *dobsu*; for example, if the observed coupling is -6 Hz, and a 1 Hz "buffer" is desired, you could set *dobsl* to -7 and *dobsu* to -5.

dwt The relative weight of each observed value. Default is 1.0. The penalty function is thus:

$$E^i_{align} = D^i_{wt}(D^i_{calc} - D^i_{obs(u,l)})^2$$

where D_{wt} may vary from one observed value to the next. Note that the default value is arbitrary, and a smaller value may be required to avoid overfitting the dipolar coupling data.[493]

dataset Each dipolar peak can be associated with a "dataset", and a separate alignment tensor will be computed for each dataset. This is generally used if there are several sets of experiments, each with a different sample or temperature, etc., that would imply a different value for the alignment tensor. By default, there is one dataset to which each observed value is assigned.

num_datasets The number of datasets in the constraint list. Default is 1.

s11,s12,s13,s22,s23 Initial values for the Cartesian components of the alignment tensor. The tensor is traceless, so S33 is calculated as -(S11+S22). In order to have the order of magnitude of the S values be roughly commensurate with coordinates in Angstroms, the alignment tensor values must be multiplied by 10^5.

gigj Product of the nuclear "g" factors for this dipolar coupling restraint. These are related to the nuclear gyromagnetic rations by $\gamma_N = g_N\beta_N/\hbar$. Common values are ^{1}H = 5.5856, ^{13}C = 1.4048, ^{15}N = -0.5663, ^{31}P = 2.2632.

dij The internuclear distance for observed dipolar coupling. If a nonzero value is given, the distance is considered to be fixed at the given value. If a *dij* value is zero, its value is computed from the structure, and it is assumed to be a variable distance. For one-bond couplings, it is usually best to treat the bond distance as "fixed" to an effective zero-point vibration value.[494]

dcut Controls printing of calculated and observed dipolar couplings. Only values where abs(dobs(u,l) - dcalc) is greater than *dcut* will be printed. Default is 0.1 Hz. Set to a negative value to print all dipolar restraint information.

freezemol If this is set to *.true.*, the molecular coordinates are not allowed to vary during dynamics or minimization: only the elements of the alignment tensor will change. This is useful to fit just an alignment tensor to a given structure. Default is *.false.*.

23.6. Residual CSA or pseudo-CSA restraints

Resonance positions in partially aligned media will be shifted from their positions in isotropic media, and this can provide information that is very similar to residual dipolar coupling constriants. This section shows how to input these sorts of restraints. The entry of the alignment tensor is done as in Section 23.5, so you must have a DIPOLE file (with an &align namelist) even if you don't have any RDC restraints. Then, if there is a CSA line in I/O redirection section, that file will be read with the following inputs:

Variables in the &csa namelist.

ncsa	Number of observed residual CSA peaks to be used as restraints.
icsa,jcsa,kcsa	Atom numbers for the csa of interest: *jcsa* is the atom whose $\Delta\sigma$ value has been measured; *icsa* and *kcsa* are two atoms bonded to it, used to define the local axis frame for the CSA tensor. See *amber12/test/pcsa/RST.csa* for examples of how to set these.
cobsl, cobsu	Limiting values for the observed residual CSA, in Hz (not ppm or ppb!). If the calculated value of $\Delta\sigma$ is less than *cobsl*, the energy penalty is proportional to $(\Delta\sigma_{calc} - \Delta\sigma_{obs,l})^2$; if it is larger than *cobsu*, the penalty is proportional to $(\Delta\sigma_{calc} - \Delta\sigma_{obs,u})^2$. Calculated values between *cobsl* and *cobsu* are not penalized. Note that *cobsl* must be less than *cobsu*.
cwt	The relative weight of each observed value. Default is 1.0. The penalty function is thus: $E^i_{csa} = C^i_{wt}(\Delta\sigma^i_{calc} - \Delta\sigma^i_{obs(u,l)})^2$ where C_{wt} may vary from one observed value to the next. Note that the default value is arbitrary, and a smaller value may be required to avoid overfitting the data.
datasetc	Each residual CSA can be associated with a "dataset", and a separate alignment tensor will be computed for each dataset. This is generally used if there are several sets of experiments, each with a different sample or temperature, etc., that would imply a different value for the alignment tensor. By default, there is one dataset to which each observed value is assigned. The tensors themselves are entered for each dataset in the DIPOLE file.
field	Magnetic field (in MHz) for the residual CSA being considered here. This is indexed from 1 to *ncsa,* and is nucleus dependent. For example, if the proton frequency is 600 MHz, then *field* for ^{13}C would be 150, and that for ^{15}N would be 60.
sigma11, sigma22, sigma12, sigma13, sigma23	Values of the CSA tensor (in ppm) for atom *icsa*, in the local coordinate frame defined by atoms *icsa*, *jcsa* and *kcsa*. See *$AMBERHOME/test/pcsa/RST.csa* for examples of how to set these.
ccut	Controls printing of calculated and observed residual CSAs. Only values where abs(cobs(u,l) - ccalc) is greater than *ccut* will be printed. Default is 0.1 Hz. Set to a negative value to print all information.

The residual CSA facility is new as of Amber 10, and has not been used as much as other parts of the NMR refinement package. You should study the example files listed above to see how things work. The residual CSA values should closely match those found by the RAMAH package (*http://www-personal.umich.edu/~hashimi/Software.html*), and testing this should be a first step in making sure you have entered the data correctly.

23.7. Preparing restraint files for Sander

Fig. 23.1 shows the general information flow for auxiliary programs that help prepare the restraint files. Once the restraint files are made, Fig. 23.2 shows a flow-chart of the general way in which *sander* refinements are carried out.

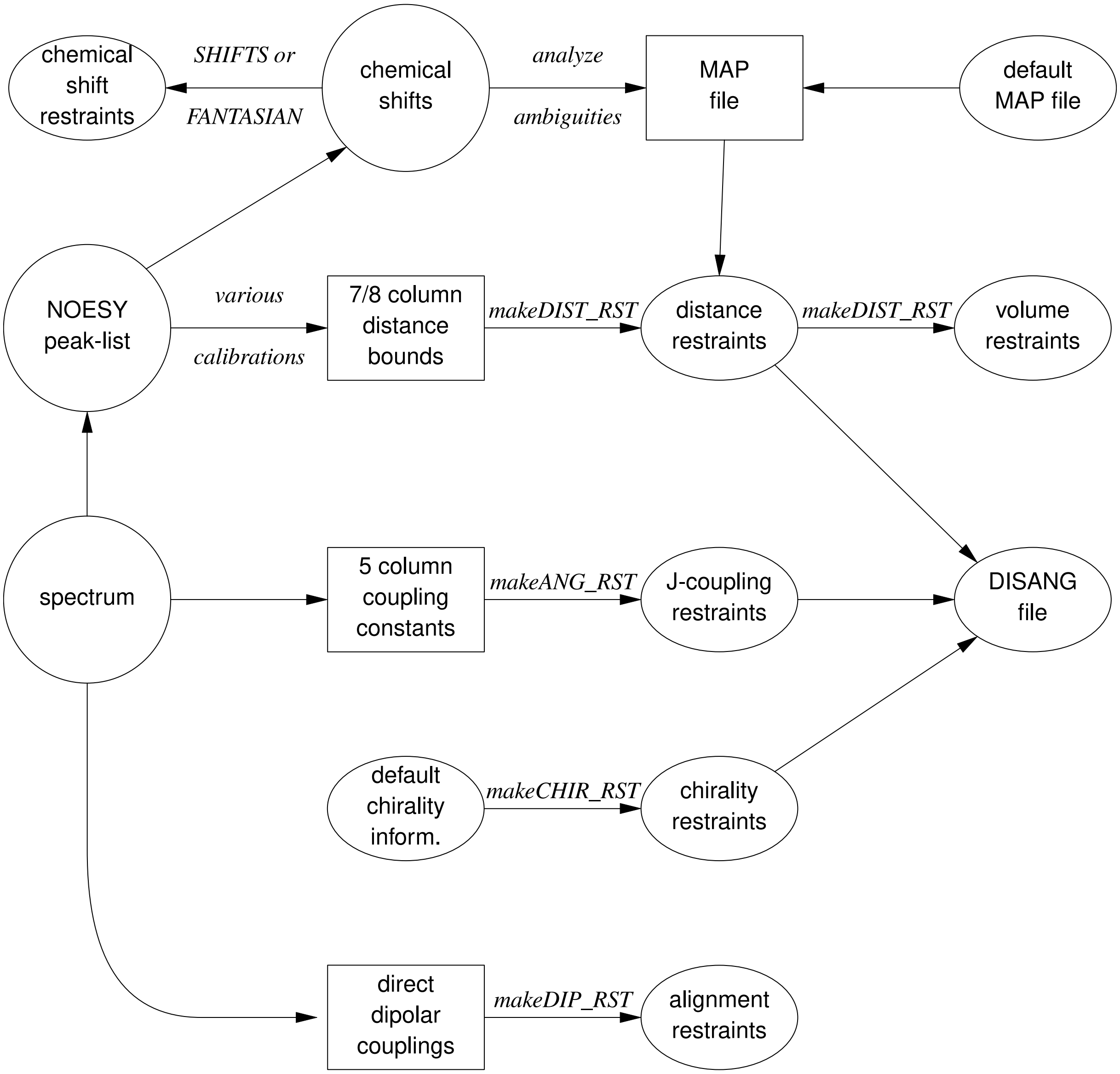

chemical shift restraints
SHIFTS or FANTASIAN
chemical shifts
analyze ambiguities
MAP file
default MAP file
NOESY peak-list
various calibrations
7/8 column distance bounds
makeDIST_RST
distance restraints
makeDIST_RST
volume restraints
spectrum
5 column coupling constants
makeANG_RST
J-coupling restraints
DISANG file
default chirality inform.
makeCHIR_RST
chirality restraints
direct dipolar couplings
makeDIP_RST
alignment restraints

Figure 23.1.: *Notation:* circles *represent logical information, whose format might differ from one project to the next;* solid rectangles *are in a specific format (largely compatible with DIANA and other programs), and are intended to be read and edited by the user;* ellipses *are specific to* sander*, and are generally not intended to be read or edited manually. The conversion of NOESY volumes to distance bounds can be carried out by a variety of programs such as* mardigras *or* xpk2bound *that are not included with Amber. Similarly, the analysis and partial assignment of ambiguous or overlapped peaks is a separate task; at TSRI, these are typically carried out using the programs* xpkasgn *and* filter.pl

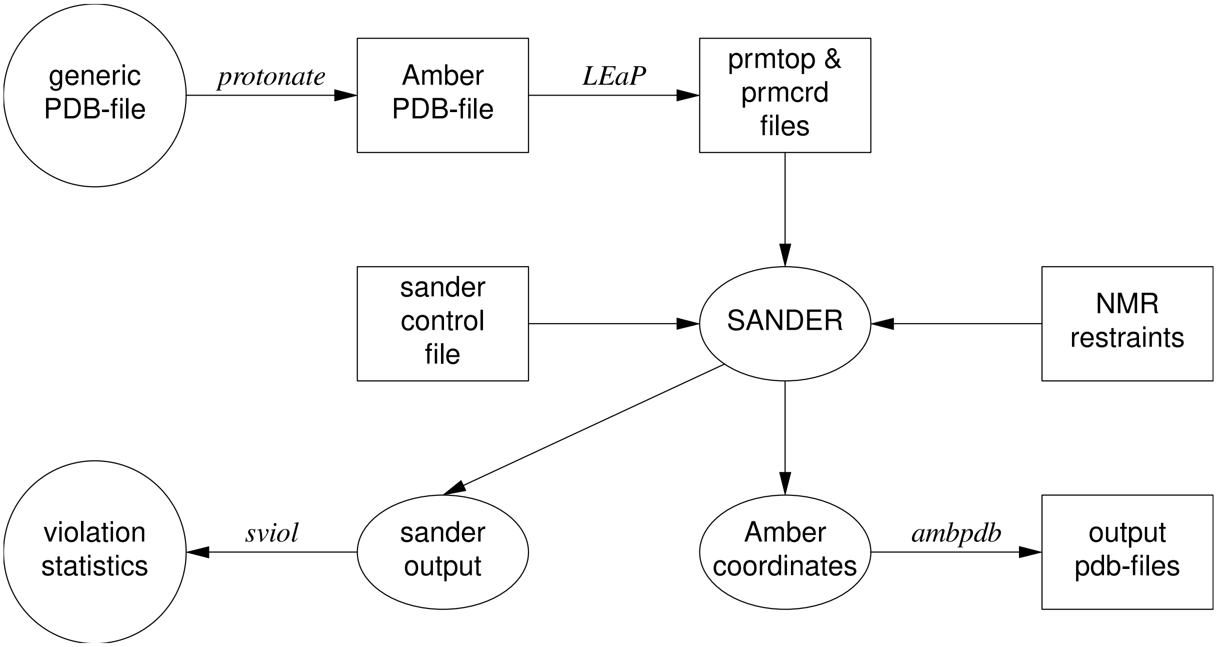

generic PDB-file
protonate
Amber PDB-file
LEaP
prmtop & prmcrd files
sander control file
SANDER
NMR restraints
violation statistics
sviol
sander output
Amber coordinates
ambpdb
output pdb-files

Figure 23.2.: *General organization of NMR refinement calculations.*

The basic ideas of this scheme owe a lot to the general experience of the NMR community over the past decade. Several papers outline procedures in the Scripps group, from which a lot of the NMR parts of *sander* are derived.[485, 495–499] 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.

23.7.1. Preparing distance restraints: makeDIST_RST

The *makeDIST_RST* program converts a simplified description of distance bounds into a detailed input for *sander*. A variety of input and output filenames may be specified on the command line:
input:

```
-upb <filename> 7-col file of upper distance bounds, OR
-ual <filename> 8-col file of upper and lower bounds, OR
-vol <filename> 7-col file of NOESY volumes
-pdb <filename> Brookhaven format file
-map <filename> MAP file (default:map.DG-AMBER)
-les <filename> LES atom mappings, made by addles
```

output:

```
-dgm <filename> DGEOM95 restraint format
-rst <filename> SANDER restraint format
-svf <filename> Sander Volume Format, for NOESY refinement
```

other options:

```
-help (gives you this explanation, overrides other parameters)
-report (gives you short runtime diagnostic output)
-nocorr (do not correct upper bound for r**-6 averaging)
-altdis (use alternative form for the distance restraints)
```

The *7/8 column distance bound* file is essentially that used by the DIANA or DISGEO programs. It consists of one-line per restraint, which would typically look like the following:

```
23 ALA HA 52 VAL H 3.8 # comments go here
```

The first three columns identify the first proton, the next three the second proton, and the seventh column gives the upper bound. Only the first three letters of the residue name are used, so that DIANA files that contain residues like "ASP-" will be correctly interpreted. An alternate, 8-column, format has both upper and lower bounds as the seventh and eighth columns, respectively. A typical line might in an "8-col" file might look like this:

```
23 ALA HA 52 VAL H 3.2 3.8 # comments go here
```

Here the lower bound is 3.2 Åand the upper bound is 3.8 Å. Comments typically identify the spectrum and peak-number or other identification that allow cross-referencing back to the appropriate spectrum. If the comment contains the pattern "<integer>:<integer>", then the first integer is treated as a peak-identifier, and the second as a spectrum-identifier. These identifiers go into the *ixpk* and *nxpk* variables, and will later be printed out in *sander*, to facilitate going back to the original spectra to track down violations, etc.

The format for the *-vol* option is the same as for the *-upb* option except that the seventh column holds a peak intensity (volume) value, rather than a distance upper bound.

The input PDB file must exactly match the Amber *prmtop* file that will be used; use the ambpdb -aatm command to create this.

If all peaks involved just single protons, and were fully assigned, this is all that one would need. In general, though, some peaks (especially methyl groups or fast-rotating aromatic rings) represent contributions from more than one proton, and many other peaks may not be fully assigned. *Sander* handles both of these situations in the same way, through the notion of an "ambiguous" peak, that may correspond to several assignments. These peaks are given two types of special names in the 7/8-column format file:

1. Commonly-occurring ambiguities, like the lack of stereospecific assignments to two methylene protons, are given names defined in the default MAP file. These names, also more-or-less consistent with DIANA, are like the names of "pseudo-atoms" that have long been used to identify such partially assigned peaks, e.g. "QB" refers to the (HB2,HB3) combination in most residues, and "MG1" in valine refers collectively to the three methyl protons at position CG1, etc.

2. There are generally also molecule-specific ambiguities, arising from potential overlap in a NOESY spectrum. Here, the user assigns a unique name to each such ambiguity or overlap, and prepares a list of the potential assignments. The names are arbitrary, but might be constructed, for example, from the chemical shifts that identify the peak, e.g. "p_2.52" might identify the set of protons that could contribute to a peak at 2.52 ppm. The chemical shift list can be used to prepare a list of potential assignments, and these lists can often be pruned by comparison to approximate or initial structures.

The default and molecule-specific MAP files are combined into a single file, which is used, along with the 7-column restraint file, the the program *makeDIST_RST* to construct the actual *sander* input files. You should consult the help file for makeDIST_RST for more information. For example, here are some lines added to the MAP file for a recent TSRI refinement:

```
AMBIG n2:68 = HE 86 HZ 86
AMBIG n2:72 = HE 24 HD 24 HZ 24
AMBIG n2:73 = HN 81 HZ 13 HE 13 HD 13 HZ 24
AMBIG n2:78 = HN 76 HZ 13 HE 13 HZ 24
AMBIG n2:83 = HN 96 HN 97 HD 97 HD 91
AMBIG n2:86 = HD1 66 HZ2 66
AMBIG n2:87 = HN 71 HH2 66 HZ3 66 HD1 66
```

Here the spectrum name and peak number were used to construct a label for each ambiguous peak. Then, an entry in the restraint file might look like this:

```
123 GLY HN 0 AMB n2:68 5.5
```

indicating a 5.5 Åupper bound between the amide proton of Gly 123 and a second proton, which might be either the HE or HZ protons of residue 86. (The "zero" residue number just serves as a placeholder, so that there will be the same number of columns as for non-ambiguous restraints.) If it is possible that the ambiguous list might not be exhaustive (e.g. if some protons have not been assigned), it is safest to set *ialtd*=1, which will allow "mistakes" to be present in the constraint list. On the other hand, if you want to be sure that every violation is "active", set *ialtd*=0.

If the *-les* flag is set, the program will prepare distance restraints for multiple copies (LES) simulations. In this case, the input PDB file is one *without* LES copies, i.e. with just a single copy of the molecule. The "lesfile" specified by this flag is created by the *addles* program, and contains a mapping from original atom numbers into the copy numbers used in the multiple-copies simulation.

The *-rst* and *-svf* flags specify outputs for *sander*, for distance restraints and NOESY restraints, respectively. In each case, you may need to hand-edit the outputs to add additional parameters. You should make it a habit to compare the outputs with the descriptions given earlier in this chapter to make sure that the restraints are what you want them to be.

It is common to run makeDIST_RST several times, with different inputs that correspond to different spectra, different mixing times, etc. It is then expected that you will manually edit the various output files to combine them into the single file required by *sander*.

23.7.2. Preparing torsion angle restraints: makeANG_RST

There are fewer "standards" for representing coupling constant information. We have followed the DIANA convention in the program *makeANG_RST*. This program takes as input a five-column torsion angle constraint file along with an Amber PDB file of the molecule. It creates as output (to standard out) a list of constraints in RST format that is readable by Amber.

```
Usage: makeANG_RST -help
makeANG_RST -pdb ambpdb_file [-con constraint] [-lib libfile]
[-les lesfile ]
```

The input torsion angle constraint file can be read from standard in or from a file specified by the -con option on the command line. The input constraint file should look something like this:

```
1 GUA PPA 111.5 144.0
2 CYT EPSILN 20.9 100.0
2 CYT PPA 115.9 134.2
3 THY ALPHA 20.4 35.6
4 ADE GAMMA 54.7 78.8
5 GLY PHI 30.5 60.3
6 ALA CHI 20.0 50.0
....
```

Lines beginning with "#" are ignored. The first column is the residue number; the second is the residue name (three letter code, or as defined in your personal torsion library file). Only the first three letters of the residue name are used, so that DIANA files that contain residues like "ASP-" will be correctly interpreted. Third is the angle name (taken from the torsion library described below).The fourth column contains the lower bound, and the fifth column specifies the upper bound. Additional material on the line is (presently) ignored.

Note: It is assumed that the lower bound and the upper bound define a region of allowed conformation on the unit circle that is swept out in a clockwise direction from $lb \rightarrow ub$. If the number in the *lb* column is greater than the the number in the *ub* column, 360°will successively be subtracted from the *lb* until $lb < ub$. This preserves the clockwise definition of the allowed conformation space, while also making the number that specifies the lower bound less than the number that specifies the upper bound, as is required by Amber. If this occurs, a warning message will be printed to *stderr* to notify the user that the data has been modified.

The angles that one can constrain in this manner are defined in the library file that can be optionally specified on the command line with the -lib flag, or the default library "tordef.lib" (written by Garry P. Gippert) will be used. If you wish to specify your own nomenclature, or add angles that are not already defined in the default file, you should make a copy of this file and modify it to suit your needs. The general format for an entry in the library is:

```
LEU PSI N CA C N+
```

where the first column is the residue name, the second column is the angle name that will appear in the input file when specifying this angle, and the last four columns are the atom names that define the torsion angle. When a torsion angle contains atom(s) from a preceding or succeeding residue in the structure, a "-" or "+" is appended to those atom names in the library, thereby specifying that this is the case. In the example above, the atoms that define PSI for LEU residues are the N, CA, and C atoms of that same LEU and the N atom of the residue after that LEU in the primary structure. Note that the order of atoms in the definition is important and should reflect that the torsion angle rotates about the two central atoms as well as the fact that the four atoms are bonded in the order that is specified in the definition.

If the first letter of the second field is "J", this torsion is assumed to be a J-coupling constraint. In that case, three additional floats are read at the end of the line, giving the A,B and C coefficients for the Karplus relation for this torsion. For example:

```
ALA JHNA H N CA HA 9.5 -1.4 0.3
```

will set up a J-coupling restraint for the HN-HA 3-bond coupling, assuming a Karplus relation with A,B, C as 9.5, -1.4 and 0.3. (These particular values are from Brüschweiler and Case, JACS 116: 11199 (1994).)

This program also supports pseudorotation phase angle constraints for prolines and nucleic acid sugars; each of these will generate restraints for the 5 component angles which correspond to the *lb* and *ub* values of the input pseudorotation constraint. In the torsion library, a pseudorotation definition looks like:

```
PSEUDO CYT PPA NU0 NU1 NU2 NU3 NU4
CYT NU0 C4' O4' C1' C2'
CYT NU1 O4' C1' C2' C3'
CYT NU2 C1' C2' C3' C4'
CYT NU3 C2' C3' C4' O4'
CYT NU4 C3' C4' O4' C1'
```

The first line describes that a PSEUDOrotation angle is to be defined for CYT that is called PPA and is made up of the five angles NU0-NU4. Then the definition for NU0-NU4 should also appear in the file in the same format as the example given above for LEU PSI.

PPA stands for Pseudorotation Phase Angle and is the angle that should appear in the input constraint file when using pseudorotation constraints. The program then uses the definition of that PPA angle in the library file to look for the 5 other angles (NU0-NU4 in this case) which it then generates restraints for. PPA for proline residues is included in the standard library as well as for the DNA nucleotides.

If the *-les* flag is set, the program will prepare torsion angle restraints for multiple copies (LES) simulations. In this case, the input PDB file is one *without* LES copies, i.e. with just a single copy of the molecule. The "lesfile" specified by this flag is created by the *addles* program, and contains a mapping from original atom numbers into the copy numbers used in the multiple-copies simulation.

Torsion angle constraints defined here cannot span two different copy sets, i.e., there cannot be some atoms of a particular torsion that are in one multiple copy set, and other atoms from the same torsion that are in other copy sets. It *is* OK to have some atoms with single copies, and others with multiple copies in the same torsion. The program will create as many duplicate torsions as there are copies.

A good alternative to interpreting J-coupling constants in terms of torsion angle restraints is to refine directly against the coupling constants themselves, using an appropriate Karplus relation. See the discussion of the variable RJCOEF, above.

23.7.3. Chirality restraints: makeCHIR_RST

```
Usage: makeCHIR_RST <pdb-file> <output-constraint-file>
```

We also find it useful to add chirality constraints and *trans*-peptide ω constraints (where appropriate) to prevent chirality inversions or peptide bond flips during the high-temperature portions of simulated annealing runs. The program *makeCHIR_RST* will create these constraints. Note that you may have to edit the output of this program to change *trans* peptide constraints to *cis*, as appropriate.

23.7.4. Direct dipolar coupling restraints: makeDIP_RST

For simulations with residual dipolar coupling restraints, the *makeDIP_RST.protein*, *makeDIP_RST.dna* and *makeDIP_RST.diana* are simple codes to prepare the input file. Use *-help* to obtain a more detailed description of the usage. For now, this code only handles backbone NH and CαH data. The header specifying values for various parameters needs to be manually added to the output of *makeDIP_RST*.

Use of residual dipolar coupling restraints is new both for Amber and for the general NMR community. Refinement against these data should be carried out with care, and the optimal values for the force constant, penalty function, and initial guesses for the alignment tensor components are still under investigation. Here are some suggestions from the experiences so far:

1. Beware of overfitting the dipolar coupling data in the expense of Amber force field energy. These dipolar coupling data are very sensitive to tiny changes in the structure. It is often possible to drastically improve the fitting by making small distortions in the backbone angles. We recommend inclusion of explicit angle restraints to enforce ideal backbone geometry, especially for those residues that have corresponding residual dipolar coupling data.

2. The initial values for the Cartesian components of the alignment tensor can influence the final structure and alignment if the structure is not fixed (ibelly = 0). For a fixed structure (ibelly = 1), these values do not matter. Therefore, the current "best" strategy is to fit the experimental data to the fixed starting structure, and use the alignment tensor[s] obtained from this fitting as the initial guesses for further refinement.

3. Amber is capable of simultaneously fitting more than one set of alignment data. This allows the use of individually obtained datasets with different alignment tensors. However, if the different sets of data have equal directions of alignment but different magnitudes, using an overall scaling factor for these data with a single alignment tensor could greatly reduce the number of fitting parameters.

4. Because the dipolar coupling splittings depend on the square root of the order parameters ($0 \leq S2 \leq 1$), these order parameters describing internal motion of individual residues are often neglected (N. Tjandra and A. Bax, *Science* **278**, 1111-1113, 1997). However, the square root of a small number can still be noticeably smaller than 1, so this may introduce undesirable errors in the calculations.

23.7.5. Using NMR exchange format (NEF) files

The NMR community, in collaboration with the worldwide PDB, is developing a common format for encoding of NMR restraints, including all of the kinds discussed above. This format is not yet finalized, but we are including here a conversion script, *nef_to_RST*, that would convert these files to *sander* format. Because this format is so new, and is still subject to revisions, care should be taken in using this script: make sure that the output files do what they should be doing. Here are the usage instructions (which you can also get by typing “nef_to_RST -help” at the command line:

```
# nef_to_RST
convert NEF restraints to Amber format
input:
   -nef <filename>: NEF file
   -pdb <filename>: PDBFILE using AMBER nomenclature and numbering
   -map <filename>: MAP file  (default:map.NEF-AMBER)
output:
   -rst <filename>:  SANDER restraint format
   -rdc <filename>:  SANDER DIP format

other options:
   -nocorr (do not correct upper bound for r**-6 averaging)
   -altdis (use alternative form for the distance restraints)
   -help   (gives you this explanation, overrides other parameters)
```

```
      -report (gives you short runtime diagnostic output)
   errors come to stderr.
```

23.7.6. fantasian

A program to evaluate magnetic anisotropy tensor parameters

```
Ivano Bertini
Depart. of Chemistry, Univ. of Florence, Florence, Italy
e-mail: bertini@risc1.lrm.fi.cnr.it
```

INPUT FILES:

Observed shifts file (pcshifts.in):

```
1st column --> residue number
2nd column --> residue name
3rd column --> proton name
4th column --> observed pseudocontact shift value
5th column --> multiplicity of the NMR signal (for example it is 3 for of a methyl group)
6th column --> relative tolerance
7th column --> relative weight
```

Amber pdb file (parm.pdb): coordinates file in PDB format. If you need to use a solution NMR family of structures you have to superimpose the structures before to use them.

OUTPUT FILES:

Observed out file (obs.out): This file is built and read by the program itself, it reports the data read from the input files.

output file (res.out): The main output file. In this file the result of the fitting is reported. Using fantasian it is possible to define an internal reference system to visualize the orientation of the tensor axes. Then in this file you can find PDB format lines (ATOM) which can be included in a PDB file to visualize the internal reference system and the tensor axes. In the main output file all the three equivalent permutations of the tensor parameters with respect to the reference system are reported. The summary of the minimum and maximum errors and that of squared errors are also reported.

Example files: in the directory example there are all the files necessary to run a fantasian calculation:

```
fantasian.com --> run file
pcshifts.in --> observed shifts file
parm.pdb --> coordinate file in PDB format
obs.out --> data read from input files
res.out --> main output file ~
```

23.8. Getting summaries of NMR violations

If you specify LISTOUT=POUT when running *sander*, the output file will contain a lot of detailed information about the remaining restraint violations at the end of the run. When running a family of structures, it can be useful to process these output files with *sviol*, which takes a list of *sander* output files on the command line, and sends a summary of energies and violations to STDOUT. If you have more than 20 or so structures to analyze, the output from *sviol* becomes unwieldy. In this case you may also wish to use *sviol2*, which prints out somewhat less detailed information, but which can be used on larger families of structures. The *senergy* script gives a more detailed view of force-field energies from a series of structures. (We thank the TSRI NMR community for helping to put these scripts together, and for providing many useful suggestions.)

23.9. Time-averaged restraints

The model of the previous sections involves the "single-average-structure" idea, and tries to fit all constraints to a single model, with minimal deviations. A generalization of this model treats distance constraints arising from from NOE crosspeaks (for example) as being the average distance determined from a trajectory, rather than as the single distance derived from an average structure.

Time-averaged bonds and angles are calculated as

$$\bar{r} = (1/C)\left\{\int_0^t e^{(t'-t)/\tau} r(t')^{-i} dt'\right\}^{-1/i} \tag{23.1}$$

where

$\bar{r}$ = time-averaged value of the internal coordinate (distance or angle)

t = the current time

τ = the exponential decay constant

$r(t')$ = the value of the internal coordinate at time t'

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

C = a normalization integral.

Time-averaged torsions are calculated as

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

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. In the first (the default),

$$\partial E/\partial x = (\partial E/\partial \bar{r})(\partial \bar{r}/\partial r(t))(\partial r(t)/\partial x) \tag{23.2}$$

(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 $r_3 < \bar{r} < r_4$,

$$E = k_3(\bar{r} - r_3)^2$$

and similarly for other ranges of $\bar{r}$.

When the second option for calculating forces is chosen (IINC = 1 on a DISAVI, ANGAVI or TORAVI card), forces are calculated as

$$\partial E/\partial x = (\partial E/\partial \bar{r})(\partial r(t)/\partial x) \tag{23.3}$$

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

$$\partial E/\partial x = 2k_3(\bar{r} - r_3)(\partial r(t)/\partial x)$$

Integration of this equation does not give Eq. 23.2, 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. 23.2], 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 Eq. 23.3 are not conservative forces, and would cause the system to gradually heat up, if no velocity rescaling were performed. The temperature coupling algorithm should act to maintain the average temperature near the target value. At any rate, this heating tendency should not be a problem in simulations, such as fitting NMR data, where MD is being used to sample conformational space rather than to extract thermodynamic data.

This section has described the methods of time-averaged restraints. For more discussion, the interested user is urged to consult studies where this method has been used.[500–504]

23.10. Multiple copies refinement using LES

NMR restraints can be made compatible with the multiple copies (LES) facility; see the following chapter for more information about LES. To use NMR constraints with LES, you need to do two things:

(1) Add a line like "file wnmr name=(lesnmr) wovr" to your input to *addles.* The filename (lesnmr in this example) may be whatever you wish. This will cause *addles* to output an additional file that is needed at the next step.

(2) Add "-les lesnmr" to the command line arguments to *makeDIST_RST*. This will read in the file created by *addles* containing information about the copies. All NMR restraints will then be interpreted as "ambiguous" restraints, so that if any of the copies satisfies the restraint, the penalty goes to zero.

Note that although this scheme has worked well on small peptide test cases, we have yet not used it extensively for larger problems. This should be treated as an experimental option, and users should use caution in applying or interpreting the results.

23.11. Some sample input files

The next few pages contain excerpts from some sample NMR refinement files used at TSRI. The first example just sets up a simple (but often effective) simulated annealing run. You may have to adjust the length, temperature maximum, etc. somewhat to fit your problem, but these values work well for many "ordinary" NMR problems.

23.11.1. 1. Simulated annealing NMR refinement

```
15ps simulated annealing protocol
&cntrl
  nstlim=15000, ntt=1, (time limit, temp. control)
  ntpr=500, pencut=0.1, (control of printout)
  ipnlty=1, nmropt=1, (NMR penalty function options)
  vlimit=10, (prevent bad temp. jumps)
  ntb=0, (non-periodic simulation)
/
&ewald
  eedmeth=5, (use r dielectric)
/
#
# Simple simulated annealing algorithm:
#
# from steps 0 to 1000: raise target temperature 10-1200K
# from steps 1000 to 3000: leave at 1200K
# from steps 3000 to 15000: re-cool to low temperatures
#
&wt type='TEMP0', istep1=0,istep2=1000,value1=10.,
  value2=1200., /
&wt type='TEMP0', istep1=1001, istep2=3000, value1=1200.,
  value2=1200.0, /
&wt type='TEMP0', istep1=3001, istep2=15000, value1=0.,
  value2=0.0, /
#
# Strength of temperature coupling:
# steps 0 to 3000: tight coupling for heating and equilibration
# steps 3000 to 11000: slow cooling phase
# steps 11000 to 13000: somewhat faster cooling
# steps 13000 to 15000: fast cooling, like a minimization
#
&wt type='TAUTP', istep1=0,istep2=3000,value1=0.2,value2=0.2, /
&wt type='TAUTP', istep1=3001,istep2=11000,value1=4.0,value2=2.0, /
&wt type='TAUTP', istep1=11001,istep2=13000,value1=1.0,value2=1.0, /
&wt type='TAUTP', istep1=13001,istep2=14000,value1=0.5,value2=0.5, /
&wt type='TAUTP', istep1=14001,istep2=15000,value1=0.05,value2=0.05, /
#
# "Ramp up" the restraints over the first 3000 steps:
#
&wt type='REST', istep1=0,istep2=3000,value1=0.1,value2=1.0, /
&wt type='REST', istep1=3001,istep2=15000,value1=1.0,value2=1.0, /
&wt type='END' /
LISTOUT=POUT (get restraint violation list)
DISANG=RST.f (file containing NMR restraints)
```

The next example just shows some parts of the actual RST file that *sander* would read. This file would ordinarily *not* be made or edited by hand; rather, run the programs *makeDIST_RST*, *makeANG_RST* and *makeCHIR_RST*, combining the three outputs together to construct the RST file.

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

```
# first, some distance constraints prepared by makeDIST_RST:
 # (comment line is input to makeRST, &rst namelist is output)
 #
 #( proton 1 proton 2 upper bound)
 #---------------------------------------------
 #
 # 2 ILE HA 3 ALA HN 4.00
 #
 &rst iat= 23, 40, r3= 4.00, r4= 4.50,
 r1 = 1.3, r2 = 1.8, rk2=0.0, rk3=32.0, ir6=1, /
 #
 # 3 ALA HA 4 GLU HN 4.00
 #
 &rst iat= 42, 50, r3= 4.00, r4= 4.50, /
 #
 # 3 ALA HN 3 ALA MB 5.50
 #
 &rst iat= 40, -1, r3= 6.22, r4= 6.72,
 igr1= 0, 0, 0, 0, igr2= 44, 45, 46, 0, /
 #
 # .......etc......
 #
 # next, some dihedral angle constraints, from makeANG_RST:
 #
 &rst iat= 213, 215, 217, 233, r1=-190.0,
 r2=-160.0, r3= -80.0, r4= -50.0, /
 &rst iat= 233, 235, 237, 249, r1=-190.0,
r2=-160.0, r3= -80.0, r4= -50.0, /
# .......etc.......
#
# next, chirality and omega constraints prepared by makeCHIR_RST:
#
#
# chirality for residue 1 atoms: CA CG HB2 HB3
&rst iat= 3 , 8 , 6 , 7 ,
r1=10., r2=60., r3=80., r4=130., rk2 = 10., rk3=10., /
#
# chirality for residue 1 atoms: CB SD HG2 HG3
&rst iat= 5 , 11 , 9 , 10 , /
#
# chirality for residue 1 atoms: N C HA CB
&rst iat= 1 , 18 , 4 , 5 , /
#
# chirality for residue 2 atoms: CA CG2 CG1 HB
&rst iat= 22 , 26 , 30 , 25 , /
#
......etc........
# trans-omega constraint for residue 2
&rst iat= 22 , 20 , 18 , 3 ,
r1=155., r2=175., r3=185., r4=205., rk2 = 80., rk3=80., /
#
# trans-omega constraint for residue 3
&rst iat= 41 , 39 , 37 , 22 , /
#
# trans-omega constraint for residue 4
&rst iat= 51 , 49 , 47 , 41 , /
#
# ......etc........
#
The next example is an input file for volume-based NOE refinement. As with the distance/angle RST
```

23.11.3. 3. Sample NOESY intensity input file

```
# A part of a NOESY intensity file:
 &noeexp
 id2o=1, (exchangeable protons removed)
 oscale=6.21e-4, (scale between exp. and calc. intensity units)
 taumet=0.04, (correlation time for methyl rotation, in ns.)
 taurot=4.2, (protein tumbling time, in ns.)
 NPEAK = 13*3, (three peaks, each with 13 mixing times)
 EMIX = 2.0E-02, 3.0E-02, 4.0E-02, 5.0E-02, 6.0E-02,
 8.0E-02, 0.1, 0.126, 0.175, 0.2, 0.25, 0.3, 0.35,
 (mixing times, in sec.)
 IHP(1,1) = 13*423, IHP(1,2) = 13*1029, IHP(1,3) = 13*421,
 (number of the first proton)
 JHP(1,1) = 78*568, JHP(1,2) = 65*1057, JHP(1,3) = 13*421,
 (number of the second proton)
 AEXP(1,1) = 5.7244, 7.6276, 7.7677, 9.3519,
 10.733, 15.348, 18.601,
 21.314, 26.999, 30.579,
 33.57, 37.23, 40.011,
 (intensities for the first cross-peak)
 AEXP(1,2) = 8.067, 11.095, 13.127, 18.316,
 22.19, 26.514, 30.748,
 39.438, 44.065, 47.336,
 54.467, 56.06, 60.113,
 AEXP(1,3) = 7.708, 13.019, 15.943, 19.374,
 25.322, 28.118, 35.118,
 40.581, 49.054, 53.083,
 56.297, 59.326, 62.174,
 /
 SUBMOL1
 RES 27 27 29 29 39 41 57 57 70 70 72 72 82 82 (residues in this submol)
END END
```

Next, we illustrate the form of the file that holds residual dipolar coupling restraints. Again, this would generally be created from a human-readable input using the program *makeDIP_RST*.

23.11.4. Residual dipolar restraints, prepared by makeDIP_RST:

```
&align
ndip=91, dcut=-1.0, gigj = 37*-3.1631, 54*7.8467,
s11=3.883, s22=53.922, s12=33.855, s13=-4.508, s23=-0.559,
id(1)=188, jd(1)=189, dobsu(1)= 6.24, dobsl(1)= 6.24,
id(2)=208, jd(2)=209, dobsu(2)= -10.39, dobsl(1)= -10.39,
id(3)=243, jd(3)=244, dobsu(3)= -8.12, dobsl(1)= -8.12,
....
id(91)=1393, jd(91)=1394, dobsu(91)= -19.64, dobsl(91) = -19.64,
/
```

Finally, we show how the detailed input to *sander* could be used to generate a more complicated restraint. Here is where the user would have to understand the details of the RST file, since there are no "canned" programs to create this sort of restraint. This illustrates, though, the potential power of the program.

23.11.5. A more complicated constraint

```
# 1) Define two centers of mass. COM1 is defined by
# {C1 in residue 2; C1 in residue 3; N2 in residue 4; C1 in residue 5}.
# COM2 is defined by {C4 in residue 1; O4 in residue 1; N* in residue 1}.
# (These definitions are effected by the igr1/igr2 and grnam1/grnam2
# variables; You can use up to 200 atoms to define a center-of-mass
# group)
#
# 2) Set up a distance restraint between COM1 and COM2 which goes from a
# target value of 5.0A to 2.5A, with a force constant of 1.0, over steps 1-5000.
#
# 3) Set up a distance restraint between COM1 and COM2 which remains fixed
# at the value of 2.5A as the force slowly constant decreases from
# 1.0 to 0.01 over steps 5001-10000.
#
# 4) Sets up no distance restraint past step 10000, so that free (unrestrained)
# dynamics takes place past this step.
#
&rst iat=-1,-1, nstep1=1,nstep2=5000,
  iresid=1,irstyp=0,ifvari=1,ninc=0,imult=0,ir6=0,ifntyp=0,
  r1=0.00000E+00,r2=5.0000,r3=5.0000, r4=99.000,rk2=1.0000,rk3=1.0000,
  r1a=0.00000E+00,r2a=2.5000,r3a=2.5000, r4a=99.000,rk2a=1.0000,rk3a=1.0000,
  igr1 = 2,3,4,5,0, grnam1(1)='C1',grnam1(2)='C1',grnam1(3)='N2',
  grnam1(4)='C1', igr2 = 1,1,1,0, grnam2(1)='C4',grnam2(2)='O4',grnam2(3)='N*',
/
&rst iat=-1,-1, nstep1=5001,nstep2=10000,
  iresid=1,irstyp=0,ifvari=1,ninc=0,imult=0,ir6=0,ifntyp=0,
  r1=0.00000E+00,r2=2.5000,r3=2.5000, r4=99.000,rk2=1.0000,rk3=1.0000,
  r1a=0.00000E+00,r2a=2.5000,r3a=2.5000, r4a=99.000,rk2a=1.0000,rk3a=0.0100,
  igr1 = 2,3,4,5,0, grnam1(1)='C1',grnam1(2)='C1',grnam1(3)='N2',
  grnam1(4)='C1', igr2 = 1,1,1,0, grnam2(1)='C4',grnam2(2)='O4',grnam2(3)='N*',
/
```

23.12. X-ray Crystallography Refinement using SANDER

An interface program links the SANDER and Crystallography and NMR System (CNS) software packages[505] to run QM/MM refinement on X-ray crystal structures, which in many instances will lead to an improvement of X-ray crystal structure quality for medium to low resolution datasets.[506, 507] The QM calculation is enabled by a linear scaling semi-empirical technique, the divide-and-conquer method, allowing large portions of a protein to be studied at a quantum mechanical level of theory while still retaining charge effects from the surrounding protein.[312, 313, 508]

SANDER computes forces to make an additional call to the interface program, where the atomic coordinates are output to a scratch file, CNS is then invoked via a system call to calculate the X-ray target function and its gradient in Cartesian space based on the coordinates in the scratch file. In practice, this is accomplished by modifying the CNS input script, minimize.inp. It does not perform minimization but only evaluates and outputs the X-ray target function and gradient based on the input structure. Next the X-ray target function and the gradient deposited in the scratch files are read into SANDER and added to the physical energy and gradient according to following equations. The QM/MM refinement proceeds by minimizing the total target function.

$$\mathrm{E}_{total} = \mathrm{E}_{chem} + w_{xray}\mathrm{E}_{xray}$$

The QM/MM setup in SANDER is discussed in Chapter 10. In order to run QM/MM refinement in Amber, you should have CNS already installed and set up the environment variables for CNS in your shell script. Make sure you can run "`cns_solve < minimize.inp > minimize.out`" directly from your working directory. Convert the PDB structure factors file into CNS format. Generate the input topology and coordinate files for the CNS refinement. If necessary, construct topology and parameter files for unusual ligands as well. The initial coordinates in the SANDER and CNS input files should be consistent with one another. Provide the necessary (and correct) information about crystal structure in the minimize.inp, such as crystal data, space group, etc. Change the weighting factor (w_{xray}) to balance QM/MM chemical data and the crystallographic data as appropriate.

File Usage

```
sander [-help] [-O] -i qmmmin -o qmmmout -p prmtop -c inpcrd -x qmmmcrd -cns
```

	Files for sander
qmmmin	Control data for the QM/MM minimization run
prmtop	Molecular topology, force field, periodic box type, atom and residue names
inprcd	Initial coordinates
	Files for CNS
minimize.inp	Modified minimization input
protein.cv	Structure factors file in cns format
xref.in	Link file connected between sander and CNS
generate.mtf	Topology file in CNS format
generate.pdb	Initial coordinates in CNS format

Sample inputs and outputs are in the *$AMBERHOME/test/1vrp_xray* directory.

23.13. EMAP restraints for rigid and flexible fitting into EM maps

EMAP restrained simulation[362, 509] was developed to incorporate electron microscopy (EM) image information into macromolecular structure determination. Different from NMR and X-ray data, EM images have low resolutions (5~50Å). However, EM images of large molecular assemblies up to millions of atoms and in various biologically relevant environments are available. These low resolution images provide precious structural information that can help to determine structures of many molecular assemblies and machineries[509–519].

With EMAP restraints, Sander and PMEMD can be used to perform both rigid[509] and flexible[362] fitting of molecules into experimental maps of complexes to obtain both complex structures and conformations agreeing with experimental maps. In addition to experimental map information, homologous structural information can be used by EMAP to perform targeted conformational search (TCS) to induce simulation systems to form structures of interest.

If the restraint map or structure is very different from the starting conformation, SGLD is recommended to induce large conformational change by setting *isgld*=1. This is often used to simulate conformational transition between different states. See the Sampling and free energy search section 20.1 for details on running SGLD.

If domain motion is desired while domain structures need to be maintained, one can use an EMAP restraint generated from the initial coordinates for each domain and set *move*=1 to allow the restraint map to move with the domain, so that domains can search the conformational space without unfolding or changing shape.

Each EMAP restraint is defined by a map file and a selection of atoms, as well as related parameters. Multiple EMAP restraints can be defined. The map can be either input from an image file, or generated from a pdb structure or derived from the starting coordinates. The definition of EMAP restraints are read in from the input file as “&emap” namelists. The following are variables in each &emap namelist.

mapfile	The filename of a restraint map or structure. The restraint maps must be in “map”, “ccp4”, or “mrc” format. The structure must be in pdb format. The structure need not be the same as the simulation system. A resolution can be specified for the conversion to a density map. When a blank filename is specified, *mapfile*=”, the input coordinates of the masked atoms will be used to generate a restraint map (default=”).
atmask	The atom mask for selecting atoms to be restrained (default=’:*’).
fcons	The restraining constant (default=0.05 kcal/g).
move	Allow the restraint map to move when *move*>0 (default=0).
resolution	The resolution used to convert an atomic structure to a map (default=2 Å).
ifit	Perform rigid fitting before simulation when *ifit*>0. One would do this when the initial coordinates don’t match those of the map (default=0). When ifit=1, the map is transformed (by translation and rotation) to match the coordinates; the coordinates are not altered. EMAP allows output of the re-oriented map (*mapfit*=...) that matches the (final) simulation coordinates, *and/or* output of the coordinates (*molfit*=...) that would match the orientation of the original map. When ifit=2, the masked atoms will be transformed to fit the map and the transformed coordinates will be used for the following simulation. For periodic systems, ifit=2 may cause atoms to clash with periodic image atoms.
grids	Grid numbers in x,y,z,phi,psi,theta dimensions for grid-threading rigid fitting[509]. For example, grids=2,2,2,3,3,3 defines 2 grid points in each of x,y,z directions between the minimum and maximum coordinates, and 3 grid points in each of phi (0-360), psi(0-360), theta(0-180) angles. A search for local minimums starts from every grid point and the global minimum is identified from all the local minimums (default=1,1,1,1,1,1).
mapfit	The filename for the final constraint map after rigid fitting and/or moving. The filename must has an extension of .map, .ccp4, or .mrc (default=″, for no map output).

molfit The filename for the final restrained atom coordinates after rigid fitting and/or simulation. The filename must have an extension of .pdb (default=″, for no structure output).

Here is an example input file for an EMAP constrained SGLD simulation:

```
Map Constraint Self-Guided Langevin dynamics
&cntrl ntx=1, ntb=0,nstlim=100000, imin=0, maxcyc=1, ntc=2, ntf=2, cut=9.0,
ntpr=1000, ntwr=100000,ntwx=10000, ntt=3, gamma_ln=10.0,nscm=100, dt=0.001,
ntb=0,igb=0,ips=1, isgld=1, tsgavg=1.0, sgft=0.5,tempsg=0,         (SGLD)
iemap=1,                   (turn on EMAP )
/
&emap          (EMAP restraint 1 )
mapfile='data/1gb1.ccp4',  (map is input from a map file)
atmask=':1-20',            (residues 1-20 are restrained)
fcons=0.1,
move=1,                    (restraint map can move)
ifit=1,                    (perform rigid fitting first)
mapfit='scratch/gb1n_1.ccp4',  (final map)
molfit='scratch/gb1n_1.pdb', / (final restrained atoms related to initial map)
&emap          (EMAP restraint 2)
mapfile='data/1gb1.pdb',  (map is generated from a pdb file)
atmask=':22-37',          (residues 22-37 are restrained)
fcons=0.1,move=0,         (restraint map is fixed)
ifit=1,                   (perform rigid fitting first)
mapfit='scratch/gb1h_1.ccp4',   (final map, same as initial)
molfit='scratch/gb1h_1.pdb', / (final restrained atoms related to initial map)
&emap          (EMAP restraint 3)
mapfile='',    (map is generated from initial coordinates)
atmask=':41-56',  (residues 41-56 are restrained)
fcons=0.1,move=1, (restraint map can move)
ifit=1,       (perform rigid fitting first)
mapfit='scratch/gb1c_1.ccp4',  (final map)
molfit='scratch/gb1c_1.pdb', / (final restrained atoms related to initial map)
```

24. LES

The LES functionality for sander was written by Carlos Simmerling. 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.

24.1. Preparing to use LES with Amber

The first decision that must be made is whether LES is an appropriate technique for the system that you are studying. For further guidance, you may wish to consult published articles to see where LES has proven useful in the past. Several examples will also be given at the end of this section in order to provide models that you may wish to follow.

There are three main issues to consider before running the ADDLES module of Amber.

1. What should be copied?
2. How many copies should be used?
3. How many regions should be defined?

A brief summary of my experience with LES follows.

1. You should make copies of flexible regions of interest. This sounds obvious, and in some cases it is. If you are interested in determining the conformation of a protein loop, copy the loop region. If you need to determine the position of a side chain in a protein after a single point mutation, copy that side chain. If the entire biomolecule needs refinement, then copy the entire molecule. Some other cases may not be obvious-you may need to decide how far away from a particular site structural changes may propagate, and how far to extend the LES region.
2. You should use as few copies as are necessary. While this doesn't sound useful, it illustrates the general point–too few copies and you won't get the full advantages of LES, and too many will not only increase your system size unnecessarily but will also flatten the energy surface to the point where minima are no longer well defined and a wide variety of structures become populated. In addition, remember that LES is an approximation, and more copies make it more approximate. Luckily, published articles that explore the sensitivity of the results to the number of copies show that 3-10 copies are usually reasonable and provide similar results, with 5 copies being a good place to start.
3. Placing the divisions between regions can be the most difficult choice when using LES. This is essentially a compromise between surface smoothing and copy independence. The most effective surface-smoothing in LES takes places between LES regions. This is because Na copies in region A interact with all Nb copies in region B, resulting in Na*Nb interactions, with each scaled by 1/(Na*Nb) compared to the original interaction. This is better both from the statistics of how many different versions of this interaction contribute to the LES average, and how much the barriers are reduced. Remember that since the copies of a given region do not interact with different copies of that same region, interactions inside a region are only scaled by 1/N.

The other thing to consider is whether these enhanced statistics are actually helpful. For example, if the copies cannot move apart, you will obtain many copies of the same conformation–obviously not very helpful. This will also result in less effective reduction in barriers, since the average energy barriers will be very similar to the non-average barrier. The independence of the copies is also related to how the copies are attached. For example, different copies of an amino acid side chain are free to rotate independently (at least within restrictions imposed by the surroundings and intrinsic potential) and therefore each side chain in the sequence could be placed into a separate LES region. If you are interested in backbone motion, however, placing each amino acid into a separate region is not the best choice. Each copy of a given amino acid will be bonded to the neighbor residues on each side. This restriction means that the copies are not very independent, since the endpoints for each copy need to be in nearly the same places. A better choice is to use regions of 2-4 amino acids. As the regions get larger, each copy can start to have more variety in conformation- for example, one segment may have some copies in a helical conformation while others are more strand-like or turn-like. The general rule is that larger regions are more independent, though you need to consider what types of motions you expect to see.

The best way to approach the division of the atoms that you wish to copy into regions is to make sure that you have several LES regions (unless you are copying a very small region such as a short loop or a small ligand). This will ensure plenty of inter-copy averaging. Larger regions permit wider variations in structure, but result in less surface smoothing. A subtle point should be addressed here- the statistical improvement available with LES is not a benefit in all cases and care must be taken in the choice of regions. For example, consider a ligand exiting a protein cavity in which a side chain acts as a *gate* and needs to move before the ligand can escape. If we make multiple copies of the gate, and do not copy the ligand, the ligand will interact in an average way with the *gates*. If the gate was so large that even the softer copies can block the exit, then the ligand would have to wait until ALL of the gate copies opened in order to exit. This may be more statistically difficult than waiting for the original, single gate to open despite the reduced barriers. Another way to envision this is to consider the ligand trying to escape against a true probability distribution of the gate- if it was open 50% of the time and closed 50%, then the exit may still be completely blocked. Continuum representations are therefore not always the best choice.

Specific examples will be given later to illustrate how these decisions can be made for a particular system.

24.2. Using the ADDLES program

The ADDLES module of Amber is used to prepare input for simulations using LES. A non-LES prmtop and prmcrd file are generated using a program such as LEaP. This prmtop file is then given to ADDLES and replaced by a new prmtop file corresponding to the LES system. All residues are left intact- copies of atoms are placed in the same residue as the original atom, so that analysis based on sequence is preserved. Atom numbering is changed, but atom names are unchanged, meaning that a given residue may have several atoms with the same name. A different program is available for taking this new topology file and splitting the copies apart into separate residues, if desired. All copies are given the same coordinates as in the input coordinate file for the non-LES system.

Using addles:

```
addles < inputfile > outputfile
```

SAMPLE INPUT FILE:

```
~ a line beginning with ~ is a comment line.
~ all commands are 4 letters.
~ the maximum line length is 80 characters;
~ a trailing hyphen, "-", is the line continuation token.
~ use 'file' to specify an input/output file, then the type of file
 'rprm' means this is the file to read the prmtop
~ the 'read' means it is an input file
~
file rprm name=(solv200.topo) read
~
~ 'rcrd' reads the original coordinates- optional, only if you want
~ a set of coords for the new topology
~ you can also use 'rcvd' for coords+velocities, 'rcvb' for coords,
~ velos and box dimensions, 'rcbd' for coords and box dimensions.
~ use "pack=n" option to read in multiple sets of coordinates and
~ assign different coordinates to different copies.
```

```
file rcrd name=(501v200.coords) read
~ 'wprm' is the new topology file to be written. the 'wovr' means to
~ write over the file if it exists, 'writ' means don't write over.
file wprm name=(lesparm) wovr
~ 'wcrd is for writing coords, it will automatically write velo and box
~ if they were read in by 'rcvd' or 'rcvb'
file wcrd name=(lescrd) wovr
~ now put 'action' before creating the subspaces
action
~ the default behavior is to scale masses by 1/N.
~ omas leaves all masses at the original values
omas
~ now we specify LES subspaces using the 'spac' keyword, followed
~ by the number of copies to make and then a pick command to tell which
~ atom to copy for this subspace
~ 3 copies of the fragment consisting of monomers (=residues) 1 and 2
spac numc=3 pick #mon 1 2 done
~ 3 copies of the fragment consisting of monomers 3 and 4
spac numc=3 pick #mon 3 4 done
~ 3 copies of the fragment consisting of residues 5 and 6
spac numc=3 pick #mon 5 6 done
~ 2 copies of the side chain on residue 1
~ note that this replaces each of the side chains ON EACH OF THE 3
~ COPIES MADE ABOVE with 2 copies - net 6 copies
~ each of the 3 copies of residue 1-2 has 2 side chain copies.
~ the '#sid' command picks all atoms in the residue except
~ C,O,CA,HA,N,H and HN.
spac numc=2 pick #sid 1 1 done
spac numc=2 pick #sid 2 2 done
spac numc=2 pick #sid 3 3 done
spac numc=2 pick #sid 4 4 done
spac numc=2 pick #sid 5 5 done
~ use the *EOD to end the input
*EOD
```

What this does: all of the force constants are scaled in the new prmtop file by 1/N for N copies, so that this scaling does not need to be done for each pair during the nonbond calculation. Charges and VDW epsilon values are also scaled. New bond, angle, torsion and atom types are created. Any of the original types that were not used are discarded. Since each LES copy should not interact with other copies of the SAME subspace, the other copies are placed in the exclusion list. If you define very large LES regions, the exclusion list will get large and you may have trouble with the fixed length for this entry in the prmtop file- currently 8 digits.

The coordinates are simply copied - that means that all of the LES copies initially occupy the same positions in space. In this setup, the potential energy should be identical to the original system- this is a good test to make sure everything is functioning properly. Do a single energy evaluation of the LES system and the original system, using the copied coordinate file. All terms should be nearly identical (to within machine precision and roundoff). With PME on non- neutral systems, all charges are slightly modified to neutralize the system. For LES, there are a different number of atoms than in the original system, and therefore this charge modification to each atom will differ from the non-LES system and electrostatic energies will not match perfectly.

IMPORTANT: After creating the LES system, the copies will all feel the same forces, and since the coordinates are identical, they will move together unless the initial velocities are different. If you are initializing velocities using INIT=3 and TEMPI>0, this is not a problem. In order to circumvent this problem, addles slightly (and randomly) modifies the copy velocities if they were read from the coordinate input file. If the keyword "nomodv" is specified, the program will leave all of the velocities in the same values as the original file. If you do not read velocities, make sure to assign an initial nonzero temperature to the system. You should think about this and change the behavior to suit your needs. In addition, the program scales the velocities by sqrt(N) for N copies to maintain the correct thermal energy (mv2), but only when the masses are scaled (not using omas option). Again, this requires some thought and you may want different behavior. Regardless of what options are used for the velocities, further equilibration should be carried out. These options are simple attempts to keep the system close to the original state.[520]

Sometimes it is critical that different copies can have different initial coordinates (NEB for example), this is why the option "pack" is added to command rcrd(rcvd,rcvb,rcbd). To use this option, user need first concatenate different coordinates into a single file, and use "pack=n" to indicate how many sets of coordinates there are in the file, like the following example:

```
file rcrd name=(input.inpcrd) pack=4 read
```

Then addles will assign coordinates averagely. For example, if 4 sets coordinates exists in input file, and 20 copies are generated, then copy 1-5 will have coordinate set 1, copy 6-10 will have coordinates set 2, and so on. Note this option can't work with multiple copy regions now.

It is important to understand that each subsequent pick command acts on the ORIGINAL particle numbers. Making a copy of a given atom number also makes copies of all copies of that atom that were already created. This was the simplest way to be able to have a hierarchical LES setup, but you can't make extra copies of part of one of the copies already made. I'm not sure why you would want to, or if it is even correct to do so, but you should be warned. Copies can be anything -spanning residues, copies of fragments already copied, non-contiguous fragments, etc. Pay attention to the order in which you make the copies, and look carefully at the output to make sure you get what you had in mind. Addles will provide a list at the end of all atoms, the original parent atom, and how many copies were made.

There are array size limits in the file SIZE.h, I apologize in advance for the poor documentation on these. Mail *carlos.simmerling@stonybrook.edu* if you have any questions or problems.

24.3. More information on the ADDLES commands and options

file:	open a file, also use one of
rcrd:	read coords from this file
rcvd:	read coords + velo from file
rcvb:	read coords, velo and box from file
wcrd:	write coords (and more if rcvd, rcvb) to file
wprm:	write new topology file
action:	start run, all of the following options must come AFTER action
nomodv:	do NOT slightly randomize the velocities of the copies
spac:	add a new subspace definition, using a pick command (see below); follow with "*numc=# pickcmd*", where # is the number of copies to make and *pickcmd* is a pick command that selects the group of atoms to copy.
omas:	leave all masses at original values (otherwise scale 1/N)
pimd:	write an prmtop file for PIMD simulation, which contains a much smaller non-bond exclusion list, atoms from other copy will not be included in this non-bond exclusion list.

Syntax for 'pick' commands

Currently, the syntax for picking atoms is somewhat limited. Simple Boolean logic is followed, but operations are carried out in order and parentheses are not allowed.

#prt A B picks the atom range from A to B by atom number

#mon A B picks the residue range from A to B by residue number

#cca A B picks the residue range from A to B by residue number, but dividing the residue between CA and C; the CO for A is included, and the CO for monomer B is not. See Simmerling and Elber, 1994 for an example of where this can be useful.

chem prtc A picks all atoms named A, case sensitive

chem mono A picks all residues named A, case sensitive

Completion wildcards are acceptable for names: H* picks H, HA, etc. Note that H*2 will select all atoms starting with H and ignore the 2.

Boolean logic:

| *or* atoms in either group are selected

& *and* atoms must be in both groups to be selected

!= *not* A != B will pick all atoms in A that are NOT in B

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

Examples:

```
pick command                             atoms selected
pick #mon 4 19 done                      all atoms in residues 4 through 19
pick #mon 1 50 & chem mono GLY done      only GLY in residues 1 to 50
pick chem mono LYS | chem mono GLU done  any GLU or LYS residue
pick #mon 1 5 != #prt 1 3 done           residues 1 to 5 but not atoms 1 to 3
```

so, a full command to add a new subspace (LES region) with 4 copies of atoms 15 to 35 is:

```
spac numc=4 pick #prt 15 35 done
```

24.4. Using the new topology/coordinate files with SANDER

These topology files are ready to use in Sander with one exception: all of the FF parameters have been scaled by 1/N for N copies. This is done to provide the energy of the new system as an average of the energies of the individual copies (note that it is an average energy or force, not the energy or force from an average copy coordinate). However, one additional correction is required for interactions between pairs of atoms in the same LES region. Sander will make these corrections for you, and this information is just to explain what is being done. For example, consider a system where you make 2 copies of a sidechain in a protein. Each charge is scaled by 1/2. For these atoms interacting with the rest of the system, each interaction is scaled by 1/2 and there are 2 such interactions. For a pair of particles inside the sub-space, however, the interaction is scaled by 1/2*1/2=1/4, and since the copies do not interact, there are only 2 such interactions and the sum does not correspond to the correct average. Therefore, the interaction must be scaled up by a factor of N. When the PME technique is requested, this simple scaling cannot be used since the entire charge set is used in the construction of the PME grid and individual charges are not used in the reciprocal space calculation. Therefore, the intra-copy energies and forces are corrected in a separate step for PME calculations. Sander will print out the number of correction interactions that need to be calculated, and very large amounts of these will make the calculation run more slowly. PME also needs to do a separate correction calculation for excluded atom pairs (atoms that should not have a nonbonded interaction, such as those that are connected by a bond). Large LES regions result in large numbers of excluded atoms, and these will result in a larger computational penalty for LES compared to non-LES simulations. For both of these reasons, it is more efficient computationally to use smaller LES regions- but see the discussion above for how region size affects simulation efficiency. These changes are included in the LES version of Sander (sander.LES). Each particle is assigned a LES 'type' (each new set of copies is a new type), and for each pair of types there is a scaling factor for the nonbond interactions between LES particles of those types. Most of the scaling factors are 1.0, but some are not - such as the diagonal terms which correspond to interactions inside a given subspace, and also off-diagonal terms where only some of the copies are in common. An example of this type is the side chain example given above- each of the 3 backbone copies has 2 sidechains, and while interactions inside the side chains need a factor of 6, interactions between the side chain and backbone need a factor of 3. This matrix of scaling factors is stored in the new topology file, along with the type for each atom, and the number of types. The changes made in sander relate to reading and using these scale factors.

24.5. Using LES with the Generalized Born solvation model

LES simulations can be performed using the GB solvent model, with some limitations. Compared to LES simulations in explicit water, using GB with LES provides several advantages. The most important is how each of the copies interacts with the solvent. With explicit water, the water is normally not copied and therefore interacts in an average way with all LES copies. This has important consequences for solvation of the copies. If the copies move apart, water cannot overlap any of them and therefore the water cavity will be that defined by the union of the space occupied by the copies. This has two consequences. First, moving the copies apart requires creation of a larger solvent cavity and therefore copies have a greater tendency to remain together, reducing the effectiveness of LES. Second, when the copies do move apart, each copy will not be individually solvated.

These effects arise because the water interacts with all of the copies; for each copy to be solvated independently of the other copies would require copying the water molecules. This is normally not a good idea, since copying all of the water would result in very significant computational expense. Copying only water near the solute would be tractable, but one would need to ensure that the copied waters did not exchange with non-LES bulk waters.

Using GB with LES largely overcomes these problems since each copy can be individually solvated with the continuum model. Thus when one copy moves, the solvation of the other copies are not affected. This results in a more reasonable solvation of each copy and also improves the independence of the copies. Of course the resulting simulations do retain all of the limitations that accompany the GB models.

The current code allows igb values of 1, 5 or 7 when using LES. Surface area calculations are not yet supported with LES. Only a single LES region is permitted for GB+LES simulations. A new namelist variable was introduced (RDT) in sander to control the compromise of speed and accuracy for GB+LES simulations. The article referenced below provides more detail on the function of this variable. RDT is the effective radii deviation threshold. When using GB+LES, non-LES atoms require multiple effective Born radii for an exact calculation. Using these multiple radii can significantly increase calculation time required for GB calculations. When the difference between the multiple radii for a non-LES atom is less than RDT, only a single effective radius will be used. A value of 0.01 has been found to provide a reasonable compromise between speed and accuracy, and is the default value. Before using this method, it is strongly recommended that the user read the article describing the derivation of the GB+LES approach.[521]

24.6. Case studies: Examples of application of LES

24.6.1. Enhanced sampling for individual functional groups: Glucose

The first example will deal with enhancing sampling for small parts of a molecule, such as individual functional groups or protein side chains. In this case we wanted to carry out separate simulations of α and β (not converting between anomers, only for conversions involving rotations about bonds) glucose, but the 5 hydroxyl groups and the strong hydrogen bonds between neighboring hydroxyls make conversion between different rotamers slow relative to affordable simulation times. The eventual goal was to carry out free energy simulations converting between anomers, but we need to ensure that each window during the Gibbs calculation would be able to sample all relevant orientations of hydroxyl groups in their proper Boltzmann-weighted populations. We were initially unsure how many different types of structures should be populated and carried out non-LES simulations starting from different conformations. We found that transitions between different conformations were separated by several hundred picoseconds, far too long to expect converged populations during each window of the free energy calculation. We therefore decided to enhance conformational sampling for each hydroxyl group by making 5 copies of each hydroxyl hydrogen and also 5 copies of the entire hydroxymethyl group. Since the hydroxyl rotamer for each copy should be relatively independent, we decided to place each group in a different LES region. This meant that each hydroxyl copy interacted with all copies of the neighboring groups, with a total of 5*5*5*5*5 or 3125 structural combinations contributing to the LES average energy at each point in time. The input file is given below.

```
file rprm name=(parm.solv.top) read
file rcvb name=(glucose.solv.equ.crd) read
file wprm name=(les.prmtop) wovr
file wcrd name=(glucose.les.crd) wovr
action
omas
~ 5 copies of each hydroxyl hydrogen- copying oxygen will make no difference
~ since they will not be able to move significantly apart anyway
spac numc=5 pick chem prtc HO1 done
spac numc=5 pick chem prtc HO2 done
spac numc=5 pick chem prtc HO3 done
spac numc=5 pick chem prtc HO4 done
~ take the entire hydroxy methyl group
spac numc=5 pick #prt 20 24 done
*EOD
```

This worked quite well, with transitions now occurring every few ps and populations that were essentially independent of initial conformation.[522]

24.6.2. Enhanced sampling for a small region: Application of LES to a nucleic acid loop

In this example, we consider a biomolecule (in this case a single RNA strand) for which part of the structure is reliable and another part is potentially less accurate. This can be the case in a number of different modeling situations, such as with homologous proteins or when the experimental data is incomplete. In this case two different structures were available for the same RNA sequence. While both structures were hairpins with a tetraloop, the loop conformations differed, and one was more accurate. We tested whether MD would be able to show that one structure was not stable and would convert to the other on an affordable timescale.

Standard MD simulations of several ns were not able to undergo any conversion between these two structures (the initial structure was always retained). Since the stem portion of the RNA was considered to be accurate, LES was only applied to the tetraloop region. In this case, both of the ends of the LES region would be attached to the same locations in space, and there was no concern about copies diffusing too far apart to re-converge to the same positions after optimization. The issues that need to be addressed once again are the number of copies to use, and how to place the LES region(s). I usually start with the simplest choices and used 5 LES copies and only a single LES region consisting of the entire loop. If each half of the loop was copied, then it might become too *crowded* with copies near the base-pair hydrogen bonds and conformational changes that required moving a base through this regions could become even more difficult (see the background section for details). Therefore, one region was chosen, and the RNA stem, counterions and solvent were not copied. The ADDLES input file is given below.

```
file rprm name=(prm.top) read
file rcvb name=(rna.crd) read
file wprm name=(les.parm) wovr
file wcrd name=(les.crd) wovr
action
omas
~ copy the UUCG loop region- residues 5 to 8.
~ pick by atom number, though #mon 5 8 would work the same way
spac numc=5 pick #prt 131 255 done
*EOD
```

Subsequent LES simulations were able to reproducibly convert from what was known to be the incorrect structure to the correct one, and stay in the correct structure in simulations that started there. Different numbers of LES copies as well as slightly changing the size of the LES region (from 4 residues to 6, extending 1 residue beyond the loop on either side) were not found to affect the results. Fewer copies still converted between structures, but on a slower timescale, consistent with the barrier heights being reduced roughly proportional to the number of copies used. See Simmerling, Miller and Kollman, 1998, for further details.

24.6.3. Improving conformational sampling in a small peptide

In this example, we were interested not just in improving sampling of small functional groups or even individual atoms, but in the entire structure of a peptide. The peptide sequence is AVPA, with ACE and NME terminal groups. Copying just the side chains might be helpful, but would not dramatically reduce the barriers to backbone conformational changes, especially in this case with so little conformational variety inherent in the Ala and Pro residues. We therefore apply LES to all atoms. If we copied the entire peptide in 1 LES regions, the copies could float apart. While this would not be a disaster, it would make it difficult to bring all of the copies back together if we were searching for the global energy minimum, as described above. We therefore use more than one LES region, and need to decide where to place the boundaries between regions. A useful rule of thumb is that regions should be at least two amino acids in size, so we pick our two regions as Ace-Ala-Val and Pro-Ala-Nme. If we make five LES copies of each region and each copy does not interact with other copies of the same regions, each half the peptide will be represented by five potentially different conformations at each point in time. In addition, since each copy interacts with all copies of the rest of the system, there are 25 different combinations of the two halves of the peptide that contribute at each point in time. This statistical improvement alone is valuable, but the corresponding barriers are also reduced by approximately the same factors. When we place the peptide in a solvent box the solvent interacts in an average way with each of the copies. The input file is given below, and all of the related files can be found in the test directory for LES.

```
~ all file names are specified at the beginning, before "action"
~ specify input prmtop
file rprm name=(prmtop) read
~ specify input coordinates, velocities and box (this is a restart file)
file rcvb name=(md.solv.crd) read
~ specify LES prmtop
file wprm name=(LES.prmtop) wovr
~ specify LES coordinates (and velocities and box since they were input)
file wcrd name=(LES.crd) wovr
~ now the action command reads the files and tells addles to
~ process commands
action
~ do not scale masses of copied particles
omas
~ divide the peptide into 2 regions.
~ use the CCA option to place the division between carbonyl and
~ alpha carbon
~ use the "or" to make sure all atoms in the terminal residues
~ are included since the CCA option places the region division at C/CA
~ and we want all of the terminal residue included on each end
~
~ make 5 copies of each half
~ "spac" defines a LES subspace (or region)
spac numc=5 pick #cca 1 3 | #mon 1 1 done
spac numc=5 pick #cca 4 6 | #mon 6 6 done
~ the following line is required at the end
*EOD
```

This example brings up several important questions:

1. Should I make LES copies before or after adding solvent? Since LEaP is used to add solvent, and LEaP will not be able to load and understand a LES structure, you must run ADDLES after you have solvated the peptide in LEaP. ADDLES should be the last step before running SANDER.

2. Which structure should be used as input to ADDLES? If you will also be carrying out non-LES simulations, then you can equilibrate the non-LES simulation and carry out any amount of production simulation desired before taking the structure and running ADDLES. At the point you may switch to only LES simulations, or continue both LES and non-LES from the same point (using different versions of SANDER). Typically I equilibrate my system without LES to ensure that it has initial stability and that everything looks OK, then switch to LES afterward. This way I separate any potential problems from incorrect LES setup from those arising from problems with the non-LES setup, such as in initial coordinates, LEaP setup, solvent box dimensions and equilibration protocols.

3. How can I analyze the resulting LES simulation? This is probably the most difficult part of using LES. With all of the extra atoms, most programs will have difficulty. For example, a given amino acid with LES will have multiple phi and psi backbone dihedral angles. There are basically two options: first, you can process your trajectory such that you obtain a single structure (non-LES). This might be just extracting one of the copies, or it might be one by taking the average of the LES copies. After that, you can proceed to traditional analysis but must keep in mind that the average structure may be non-physical and may not represent any actual structure being sampled by the copies, especially if they move apart significantly. A better way is to use LES-friendly analysis tools, such as those developed in the group of Carlos Simmerling. The visualization program MOIL-View (*http://morita.chem.sunysb.edu/carlos/moil-view.html*) is one example of these programs, and has many analysis tools that are fully LES compatible. Read the program web page or manual for more details.

1.7. Unresolved issues with LES in Amber

1. Sander can't currently maintain groups of particles at different temperatures (important for dynamics, less so for optimization.)[523, 524] Users can set *temp0les* to maintain all LES atoms at a temperature that is different from that for the system as a whole, but all LES atoms are then coupled to the same bath.

2. Initial velocity issues as mentioned above- works properly, user must be careful.

3. Analysis programs may not be compatible. See *http://morita.chem.sunysb.edu/carlos/moil- view.htm*l for an LES-friendly analysis and visualization program.

4. Visualization can be difficult, especially with programs that use distance-based algorithms to determine bonds. See #3 above.

5. Water should not be copied- the fast water routines have not been modified. For most users this won't matter.

6. Copies should not span different 'molecules' for pressure coupling and periodic imaging issues. Copies of an entire 'molecule' should result in the copies being placed in new, separate molecules- currently this is not done. This would include copying things such as counterions and entire protein or nucleic acid chains.

7. Copies are placed into the same residue as the original atoms- this can make some residues much larger than others, and may result in less efficient parallelization with algorithms that assign nonbond workload based on residue numbers.

25. Quantum dynamics

25.1. Path-Integral Molecular Dynamics

25.1.1. General theory

Based on Feynman's formulation of quantum statistical mechanics in terms of path-integrals, Path-Integral Molecular Dynamics (PIMD) is a computationally efficient method for calculating equilibrium (e.g., thermodynamic and structural) properties of a quantum many-body system. In the following we will briefly illustrate the basic principles, and we will derive the fundamental equations underlying its implementation using standard molecular dynamics methods. The literature should be read for a more rigorous description[249–251].

For the sake of simplicity, we restrict ourselves to a PIMD formulation of the canonical (NVT) ensemble, and we will consider a single quantum particle of mass m, with momentum p and coordinate x, which moves in a one-dimensional potential $v(x)$. Generalization to other ensembles and/or to multidimensional many-particle systems is straightforward.

In the NVT ensemble, the canonical partition function Z is expressed as

$$Z = \sum_i e^{-\beta E_i} \tag{25.1}$$

where $\beta = 1/k_B T$, and the corresponding density matrix is defined as

$$\rho = \frac{e^{-\beta E_i}}{Z} \tag{25.2}$$

The expectation value of any operator A can thus be computed as

$$\langle A \rangle = Tr(\rho A) = \frac{1}{Z} Tr\left(A e^{-\beta H}\right) \tag{25.3}$$

with H being the Hamiltonian for the one-dimensional system:

$$H = \frac{p^2}{2m} + v(x) = T + V \tag{25.4}$$

In Eq. (25.4) T and V are the kinetic and potential operators, respectively. Using the coordinate basis set $\{|\, x\rangle\}$, the canonical partition function can be computed as

$$Z = \int dx \langle x \mid e^{-\beta H} \mid x \rangle = \int dx \langle x \mid e^{-\beta (T+V)} \mid x \rangle \tag{25.5}$$

In general T and V do not commute, i.e. $[T,V] \neq 0$, and consequently $e^{-\beta(T+V)}$cannot be calculated directly. However, using he Trotter formula [525] it is possible to demonstrate that

$$Z = \lim_{P\to\infty} \int dx \langle x \mid \left(e^{-\frac{\beta V}{2P}} e^{-\frac{\beta A}{P}} e^{-\frac{\beta V}{2P}} \right)^P \mid x \rangle \tag{25.6}$$

After some algebra and using the completeness of the coordinate basis, the quantum canonical partition function can be written as

$$Z = \lim_{P\to\infty} \int dx_1 dx_2 \ldots dx_P \left(\frac{mP}{\hbar^2 \beta} \right)^{\frac{P}{2}} e^{-\sum_{i=1}^{P} \left[\frac{mP}{\beta \hbar^2} (x_{i+1} - x_i)^2 + \frac{\beta}{P} v(x_i) \right]_{x_{P+1} = x_1}} \tag{25.7}$$

Defining a "chain" frequency $\omega_P = \frac{\sqrt{P}}{\beta\hbar}$and an effective potential as

$$U_{eff}(x_1,\ldots,x_P) = \sum_{i=1}^{P}\left[\frac{1}{2}m\omega_P^2\left(x_{i+1}-x_i\right)^2 + \frac{\beta}{P}v\left(x_i\right)\right]_{x_{P+1}=x_1} \tag{25.8}$$

the canonical partition function is finally expressed as

$$Z = \lim_{P\to\infty}\int dx_1 dx_2 \ldots dx_P \left(\frac{mP}{\hbar^2\beta}\right)^{\frac{P}{2}} e^{-\beta U_{eff}(x_1,\ldots,x_P)} \tag{25.9}$$

In this form, the quantum partition function is isomorphic with a classical configurational partition function for a P-particle systems, where the P particles (generally referred to as "beads") are discrete points along a cyclic path[526]. Each bead is coupled to its nearest neighbors by harmonic springs with frequency ω_P , and is subject to the external potential *v(x)*. It is possible to make the connection between the quantum partition function and a fictitious classical P-particle system even more manifest by introducing a set of P Gaussian integrals:

$$Z = \lim_{P\to\infty}\Lambda\int dp_1 dp_2 \ldots dp_P \int dx_1 dx_2 \ldots dx_P \left(\frac{mP}{\hbar^2\beta}\right)^{\frac{P}{2}} e^{-\beta\left[\sum_{i=1}^{P}\frac{p_i^2}{2\mu_i}+U_{eff}(x_1,\ldots,x_P)\right]} \tag{25.10}$$

The new Gaussian variables are regarded as fictitious classical "momenta" and, consequently, the constants μ_i have units of mass and are generally referred to as fictitious masses. Since these Gaussian integrals are uncoupled and can be calculated analytically, the overall constant Λ can be chosen so as to reproduce the correct prefactor. Therefore, one has complete freedom to choose μ_i.

>From Eq. (25.5) it follows that the quantum partition function can be evaluated using classical molecular dynamics based on equations of motion derived from a fictitious classical Hamiltonian of the form

$$H(p,x) = \sum_{i=1}^{P}\frac{p_i^2}{2\mu_i} + U_{eff}(x_1,\ldots,x_P) \tag{25.11}$$

However, ordinary MD generates a microcanonical distribution of H, i.e., a distribution function of the form $\delta\left(H\left(p,x\right)-E\right)$, where E is the conserved energy. This is clearly not the form appearing in the quantum partition function that requires a canonical distribution of the form $e^{\beta H}$. In order to satisfy this condition, the system has to be coupled to a thermostat which guarantees that the canonical distribution is rigorously obtained.

As shown above, the exact quantum partition function is obtained in the limit of an infinite number of beads P. In practice this is obviously not possible, and therefore P must be chosen large enough that all thermodynamic properties are converged. Since P is directly related to the quantum nature of the system under consideration, a larger number of beads is necessary for systems containing light atoms (e.g., hydrogen and deuterium) and for simulations at low temperatures.

Two different implementations of PIMD are currently available in Amber. The first one corresponds to the so-called primitive approximation (PRIMPIMD) [527] which is directly obtained from the formulation provided above with the fictitious mass of each bead chosen as $\mu_i = m/P$, where m is the particle mass. In PRIMPIMD, the canonical distribution is obtained by either using a Langevin thermostat or Nosé-Hoover chains of thermostats coupled to each degree of freedom of the system according to the algorithm of Ref. [528]. The latter is the recommended option. The second implementation, which is called Normal Mode Path-Integral Molecular Dynamics (NMPIMD) [529], makes use of a normal mode transformation that uncouples the harmonic term in Eq. (25.8). As a consequence the fictitious masses are different. In the current implementation of NMPIMD, the canonical distribution is obtained by using Nosé-Hoover chains of thermostats coupled to each degree of freedom of the system. We note here that NMPIMD is preferred over PRIMPIMD because it guarantees a more efficient sampling of the phase space.

In both PRIMPIMD and NMPIMD, the equations of motion are propagated using the Leapfrog algorithm, and the quantum energies of the system (total, kinetic and potential energy) are computed using the so-called "virial estimator"[527, 530].

All the force fields available for regular MD in Amber can also be used for PRIMPIMD and NMPIMD simulations. However, we note here that the common empirical force fields may require an additional re-parameterization (see Ref. [85] for a more detailed discussion). A simple charge, flexible water model specifically developed in Ref. [85] for investigating nuclear quantum effects is already implemented in the current version of Amber (see Section 3.5) and it is recommended for PRIMPIMD and NMPIMD simulations of aqueous systems.

25.1.2. How PIMD works in Amber

Implementation and input/output files

The current implementation of PRIMPIMD and NMPIMD allows the "quantization" of either the whole system or just a part of it. In both cases the *mdin* input is the same as for a regular (classical MD) run. However, additional flags are required, which will be described in Section 25.1.2.

For cases where the whole system is quantized, the most efficient way to perform PRIMPIMD and NMPIMD simulations is with *sander.MPI* exploiting the multisander scheme. You must use the same *prmtop* file as in the corresponding classical simulation, while *P* separate coordinate files (one for each of the *P* beads) are required. The number of beads to get converged results for typical systems at ambient conditions vary between 16 and 32. However, other aspects of quantum behavior may be observed with fewer beads. Therefore, some experimentation on your system may be required to find the optimal number. In order to run the simulation you also need a multisander groupfile containing (per line) all the options for each sander job. As output, *sander.MPI* generates the same files as a regular (classical MD) run. The only difference is that there are now *P* of such files, one for each bead. Therefore, you will have *P mdout* files with the bead contributions to the quantum energies, *P rst* files with the coordinates of each bead for restart, and *P* trajectory files (*mdcrd* and *mdvel*) with the bead coordinates and velocities saved during the run. It is important to note that for both PRIMPIMD and NMPIMD the velocities do not correspond to the real-time velocities of the system but are just fictitious velocities needed to solve the integral in Eq. (25.5). *sander.MPI* also writes a general *pimdout* file, which reports the quantum results for the whole system (i.e., total, kinetic and potential energy, pressure, volume, density...). If Nosé-Hoover chains of thermostats are employed, an additional file (*NHC.dat*) is printed with the conserved energy for the extended system. You must carefully check that the timestep used in the simulation is small enough to guarantee conservation of this quantity.

For cases where only a part of the system is quantized, both PRIMPIMD and NMPIMD are implemented within the *LES* scheme (see Chapter 24). Therefore, you must use either *sander.LES* or *sander.LES.MPI,* and prepare the *prmtop* file in a special way. The input files are generated using *addles*. Basically, regular topology and coordinate files are needed, then a control script (usually named *addles.in*) should be written. The necessary input files can then be generated by running "*addles* < *addles.in*". The following is what a typical *addles.in* will look like (lines start with a "~" are comments):

```
~ designate regular topology file
file rprm name=(input.prmtop) read
~ designate normal coordinate file
file rcrd name=(input.inpcrd) read
~ where to put PIMD topology file
file wprm name=(pimd.prmtop) wovr
~ where to put PIMD coordinate file
file wcrd name=(pimd.inpcrd) wovr
action
~ use original mass(it is required by PIMD)
omas
~ make 4 copies of atom 1-648(should be the whole system)
space numc=4 pick #prt 1 648 done
*EOD
```

Several things should be emphasized here about writing *addles.in* for PRIMPIMD and NMPIMD:

1. If copies of the whole system are made, it means that the whole system is quantized. In this case, *sander.MPI* offers a more efficient way to perform PRIMPIMD and NMPIMD simulations without using *LES* (see above). We note here, that *sander.LES* (and *sander.LES.MPI)* should be used when you are interested in quantizing only a part of your system.

2. The current implementation requires that the “omas” tag must be turned on to make every atom use original mass during the simulation.
3. As mentioned above, how many copies to create is a tradeoff between accuracy and efficiency. To get converged total energies, 16-32 copies may be required; however, other aspects of quantum behavior may be seen with fewer copies. Be prepared to experiment on your system to see what is required.

As output, *sander.LES* (and *sander.LES.MPI*) generates the same files as a regular (classical MD) run. The *mdout* file contains the quantum results for the whole system (i.e., total, kinetic and potential energy, pressure, volume, density...). while the *rst* file contains the coordinates of all beads for restart. The trajectory files (*mdcrd* and *mdvel*) contain the coordinates and velocities of all beads saved during the run. If Nosé-Hoover chains of thermostats are employed, an additional file (*NHC.dat*) is printed with the conserved energy for the extended system. You must carefully check that the timestep used in the simulation is small enough to guarantee conservation of this quantity.

Input parameters

In order to perform PRIMPIMD and NMPIMD simulations, an additional flag is required in the *mdin* file, which distinguishes among the different methodologies based on the path-integral formalism.

ipimd Flag for the different methodologies based on the path-integral formalism. See Sections 25.2.1 and 25.3.1 for the other values.

= 0 defines regular MD (default).

= 1 defines PRIMPIMD.

= 2 defines NMPIMD.

As described above, in order to guarantee a proper canonical sampling of the phase space the quantum system must be coupled to a thermostat. In the current implementation, two schemes are available: Langevin thermostat and Nosé-Hoover chains of thermostats coupled to each degree of freedom of the system. As for any regular MD run, the flag that activates the thermostat is *ntt*. A Langevin thermostat is switched on using *ntt=3*, and defining a collision frequency. To activate the Nosé-Hoover chains of thermostats, you must specify *ntt=4* and provide the number of thermostats (*nchain*) in each chain. Use of Nosé-Hoover chains of thermostats is rcommended and is the only option currently available for NMPIMD (*ipimd=2*). The choice of an appropriate number of chains depends on the system. Typically, 4 thermostats (*nchain=4*) are sufficient to guarantee an efficient sampling of the phase space.

In summary:

ntt Switch for temperature scaling. See Section 17.6.7 for other options.

= 3 defines a Langevin thermostat and also requires the definition of *gamma_ln*. Available for PRIMPIMD (*ipimd=1*) only.

= 4 defines Nosé-Hoover chains of thermostats. Available for PRIMPIMD (*ipimd=1*) and NMPIMD (*ipimd=2*). It also requires the number of thermostats in a chain (*nchain*).

nchain = 2-8 number of thermostats in each Nosé-Hoover chain of thermostats (default 2, recommended ≥ 4).

Quantum simulations in the isothermic-isobaric (NPT) ensemble are possible only for NMPIMD (*ipimd=2*) and for rectangular periodic boundary conditions (*ntb=2*) with isotropic position scaling (*ntp=1*). All the other flags are identical to those for a classical MD simulation. The current implementation of NMPIMD for the NPT ensemble is based on the derivation of Ref [531].

Examples

In the following examples of input files for PRIMPIMD and NMPIMD are shown. You are also encouraged to check the test cases in $AMBERHOME/test/PIMD.

a) PRIMPIMD input for sander.LES. No periodic boundary conditions.

Test: $AMBERHOME/test/PIMD/part_pimd_water.

```
ipimd = 1     ! PRIMPIMD
ntb = 0
ntx = 1, irest = 0
cut = 100.
temp0 = 300., tempi = 300., temp0les = -1.
ntt = 3, gamma_ln = 20.     ! Langevin thermostat
dt = 0.0001, nstlim = 1000
ntpr = 100, ntwr = 1000, ntwx = 100
```

b) PRIMPIMD input for sander.LES. NVT simulation for water with only the hydrogen atoms being quantized.

Test: $AMBERHOME/test/PIMD/part_pimd_spcfw.

```
ipimd = 1     ! PRIMPIMD
ntx = 5, irest = 0
temp0 = 300., tempi = 300., temp0les = -1.
dt = 0.0002, nstlim  10
cut = 7.
ntt = 3, gamma_ln = 20.     ! Langevin thermostat
ntpr = 1, ntwr = 10, ntwx = 1
```

c) NMPIMD input for sander.LES. NPT simulation for liquid butane.

Test: $AMBERHOME/test/PIMD/part_nmpimd_ntp.

```
ipimd = 2     ! NMPIMD
ntb = 2, ntp = 1     ! isotropic position scaling
ntx = 5, irest = 0
cut = 8.
temp0 = 80., tempi = 80., temp0les = -1.
ntt = 4, nchain = 4.     ! Nose'-Hoover chains
dt = 0.0002, nstlim = 50
ntpr = 5, ntwr = 50, ntwx = 1
```

d) NMPIMD input for sander.MPI. NPT simulation for liquid water.

Test: $AMBERHOME/test/PIMD/full_pimd_ntp_water.

```
ipimd = 2     ! NMPIMD
ntb = 2, ntp = 1     ! isotropic position scaling
ntx = 5, irest = 1
cut = 7.
temp0 = 298.15
ntt = 4, nchain = 4.     ! Nose'-Hoover chains
dt = 0.0002, nstlim = 10
ntpr = 1, ntwr = 10, ntwx = 5
```

25.2. Centroid Molecular Dynamics (CMD)

Two methods based on the path-integral formalism are available to perform approximate quantum dynamical calculations: Centroid Molecular Dynamics (CMD) [532] and Ring Polymer Molecular Dynamics (RPMD) [533].

The CMD method developed by Voth and coworkers draws upon the prescription of quantum distribution functions, in which the exact quantum expressions are cast into a phase space representation leading to a classical-like physical interpretation of the variables of interest. In particular, an approximate quantum dynamics is obtained by propagating the centroid variables (i.e., positions and velocities of the center of mass of the bead polymer) according to classical-like equations of motion. The current implementation is the so-called Adiabatic CMD [534] that makes use of a normal mode representation of path-integrals where the fictitious mass of the zero-frequency mode (i.e., the centroid of the bead polymer) is given the actual mass of the atom and, contrary to NMPIMD, the fictitious masses of all the nonzero frequency modes are scaled by an adiabaticity parameter, $\gamma < 0$. This procedure decouples the centroid motion from that of the other normal modes in the same spirit of the Car-Parrinello method. Although the centroid variables move following Newton's equations of motion, Nosé-Hoover chains of thermostats must be attached to each nonzero frequency normal mode. The user is strongly encouraged to refer to the relevant literature (Ref. [532] and references therein) for a more rigorous derivation of the CMD method.

The RPMD method developed by Manolopoulos and coworkers is based on primitive PIMD. However, there are two fundamental differences: 1) each bead is given a fictitious mass equal to the actual mass of the atom (i.e., $\mu = m$), 2) the dynamics of the system is strictly determined by the fictitious Hamiltonian of Eq. (25.11), i.e., no thermostats are employed. Also in this case, the user is strongly encouraged to refer to the relevant literature for a detailed derivation of this method [533].

Both CMD and RPMD simulations provide an efficient route for the calculation of approximate Kubo transformed correlation functions, which can then be related to the true quantum correlation functions. Importantly, running several independent trajectories is required for both CMD and RPMD to guarantee a proper canonical average of the initial conditions and, consequently, to obtain converged results (see Refs. [85] and[535] for examples of CMD and RPMD simulations, respectively).

All the force fields available for regular MD simulations in Amber can be used for CMD and RPMD. However, we also note here that the common empirical force fields may require an additional reparameterization (see Ref. [85] for a more detailed discussion). A simple charge, flexible water model specifically developed in Ref. [85] for investigating nuclear quantum effects is already implemented in the current version of Amber (see Section 3.5) and it is recommended for CMD and RPMD simulations of aqueous systems.

25.2.1. Implementation and input/output files

The implementation of CMD and the input/output files are identical to those of NMPIMD (see Section 25.1.2), with few differences. In addition to the NMPIMD output files, two other files are generated for CMD: *CMD_position.dat* containing the centroid positions, and *CMD_velocity.dat* containing the centroid velocities saved along the trajectory. The format of these files is identical to that of a classical MD simulation (*mdcrd* and *mdvel*), and the frequency with which these information are saved is determined by *ntpr*. The *mdin* input is the same as for a regular (classical MD) run. However, additional flags are required as described below.

In order to perform CMD simulations the following flags are required in the *mdin* file:

ipimd Flag for the different methodologies based on the path-integral formalism. See Section 25.1.2 for the other values.

= 3 defines CMD.

adiab_param This defines the so-called adiabaticity parameter (γ) used to make the fictitious masses of the nonzero frequency normal modes small enough to decouple their motion from that of the centroid. It has been shown that $\gamma \leq 1/2P$ (where P is the total number of beads) is sufficiently small to get converged results. As a consequence of this, a smaller timestep is required. During the equilibration run (see below) you must carefully check that the timestep employed is small enough to guarantee the energy conservation of the extended system reported in the *NHC.dat* file (see Section 25.1.2). Default is 1, but you need to specify this, since default value is not appropriate.

ntt Switch for temperature scaling.

= 4 defines Nosé-Hoover chains of thermostats. It also requires the number of thermostats in a chain (*nchain*). For CMD, Nosé-Hoover chains of thermostats must be attached to each nonzero frequency normal mode.

nchain = 2-8 number of thermostats in each Nosé-Hoover chain of thermostats (default 2, recommended ≥ 4).

eq_cmd This flag must be used during the CMD equilibration to generate a canonical distribution of the centroid variables before an actual CMD run. Default is .false.

restart_cmd Flag necessary for restarting a CMD simulation.

In order to run a CMD simulation, you must first generate an equilibrated quantum configuration of your system using NMPIMD (see Section 25.1.2). A canonical distribution of the centroid variables must then be obtained from a CMD simulation with the *equilib_cmd* flag on. After this equilibration, the final configuration is then used as initial configuration for the actual CMD simulation. For restarting a CMD run the *restart_cmd* flag in the *mdin* file is required.

Importantly, for CMD simulations it is necessary that *ntb=1,* which is the default value.

25.2.2. Examples

In the following examples of input files for CMD are shown. You are also encouraged to check the test cases in $AMBERHOME/test/PIMD.

a) CMD for sander.LES. Equilibration of the centroid variables.

Test: $AMBERHOME/test/PIMD/part_cmd_water/equilib.

```
ipimd = 3         ! CMD
ntx = 5, irest = 0
ntb = 1
temp0 = 298.15, tempi = 298.15, temp0les = -1.
cut = 7.0
ntt = 4, nchain = 4.       ! Nose'-Hoover chains
dt = 0.00005, nstlim = 100
eq_cmd = .true.       ! equilibration for CMD
adiab_param = 0.5     ! adiabaticity parameter
ntpr = 20, ntwr = 100
```

b) CMD input for sander.LES. Start of an actual CMD simulation after equilibration.

Test: $AMBERHOME/test/PIMD/part_cmd_water/start.

```
ipimd = 3         ! CMD
ntx = 5, irest = 1
ntb = 1
temp0 = 298.15, tempi = 298.15, temp0les = -1.
cut = 7.0
ntt = 4, nchain = 4.       ! Nose'-Hoover chains
dt = 0.00005, nstlim = 100
eq_cmd = .false.       ! actual CMD
adiab_param = 0.5      ! adiabaticity parameter
ntpr = 20, ntwr = 100
```

c) CMD input for sander.LES. Restart of an actual CMD.

Test: $AMBERHOME/test/PIMD/part_cmd_water/restart.

```
ipimd = 3         ! CMD
ntx = 5, irest = 1
ntb = 1
temp0 = 298.15, tempi = 298.15, temp0les = -1.
cut = 7.0
```

```
ntt = 4, nchain = 4.        ! Nose'-Hoover chains
dt = 0.00005, nstlim = 100
eq_cmd = .false.        ! actual CMD
restart_cmd = .true.    ! restart
adiab_param = 0.5       ! adiabaticity parameter
ntpr = 20, ntwr = 100
```

25.3. Ring Polymer Molecular Dynamics (RPMD)

The implementation of RPMD and the necessary input/output files are identical to those of PRIMPIMD (see Section 25.1.2). The *mdin* input is the same as for a PRIMPIMD with only few differences described below.

25.3.1. Input parameters

In order to perform RPMD the following flags are required in the *mdin* file:

ipimd	Flag for the different methodologies based on the path-integral formalism. See Section 25.1.2 for the other values. **= 4** defines RPMD.
ntt	Set this to 0, for constant energy dynamics
nscm	Set this to 0, to avoid removing translational and rotational center-of-mass motion.

You must first generate an equilibrated quantum configuration of your system using PRIMPIMD (see Section 25.1.2), which is then used as initial configuration for the actual RPMD simulation.

25.3.2. Examples

In the following examples of input files for RPMD are shown. You are also encouraged to check the test cases in $AMBERHOME/test/PIMD.

a) RPMD input for sander.LES.

Test: $AMBERHOME/test/PIMD/part_rpmd_water.

```
ipimd = 4           ! RPMD
ntx = 5, irest = 0
ntt = 0
nscm = 0
temp0 = 300., temp0les = -1.
cut = 7.0
dt = 0.0002, nstlim = 10
ntpr = 1, ntwr = 10, ntwx = 1, ntwv = 1
```

25.4. Linearized semiclassical initial value representation

25.4.1. Experimental observables and thermal correlation functions

Most quantities of interest in the dynamics of complex systems can be expressed in terms of thermal time autocorrelation functions [536], which are of the form

$$C_{AB}(t) = \frac{1}{Z}\mathrm{Tr}\left(\hat{A}^{\beta} e^{i\hat{H}t/\hbar}\hat{B}e^{-i\hat{H}t/\hbar}\right) \tag{25.12}$$

where $\hat{A}^{\beta}_{std} = e^{-\beta\hat{H}}\hat{A}$ for the standard version of the correlation function, or $\hat{A}^{\beta}_{sym} = e^{-\beta\hat{H}/2}\hat{A}e^{-\beta\hat{H}/2}$ for the symmetrized version [537], or

$$\hat{A}^{\beta}_{Kubo} = \frac{1}{\beta}\int_0^{\beta} \mathrm{d}\lambda\, e^{-(\beta-\lambda)\hat{H}}\hat{A}e^{-\lambda\hat{H}} \tag{25.13}$$

for the Kubo-transformed version [538]. These three versions are related to one another by the following identities between their Fourier transforms,

$$\frac{\beta\hbar\omega}{1-e^{-\beta\hbar\omega}} I^{Kubo}_{AB}(\omega) = I^{std}_{AB}(\omega) = e^{\beta\hbar\omega/2} I^{sym}_{AB}(\omega) \tag{25.14}$$

where

$$I_{AB}(\omega) = \int\limits_{-\infty}^{\infty} dt\; e^{-i\omega t} C_{AB}(t) \tag{25.15}$$

is the partition function and $\hat{H}$ the (time-independent) Hamiltonian of the system, and $\hat{A}$ and $\hat{B}$ are operators relevant to the specific property of interest.

25.4.2. Linearized semiclassical initial value representation

The Semiclassical initial value representation (SC-IVR) approximates the forward (backward) time evolution operator $e^{-i\hat{H}t/\hbar}$ ($e^{i\hat{H}t/\hbar}$) by a phase space average over the initial conditions of forward (backward) classical trajectories [539, 540]. By making the approximation that the dominant contribution to the phase space averages comes from forward and backward trajectories that are infinitesimally close to one another, and then linearizing the difference between the forward and backward actions (and other quantities in the integrand), Miller and coworkers [541, 542] obtained the LSC-IVR, or classical Wigner model for the correlation function in Eq. (25.12),

$$C^{LSC-IVR}_{AB}(t) = Z^{-1}(2\pi\hbar)^{-N}\int d\mathbf{x}_0 \int d\mathbf{p}_0 A^{\beta}_{w}(\mathbf{x}_0,\mathbf{p}_0)\, B_w(\mathbf{x}_t,\mathbf{p}_t) \tag{25.16}$$

where A^{β}_{w} and B_w are the Wigner functions corresponding to these operators,

$$O_w(\mathbf{x},\mathbf{p}) = \int d\mathbf{x}\,\langle \mathbf{x}-\Delta\mathbf{x}/2|\,\hat{O}\,|\mathbf{x}+\Delta\mathbf{x}/2\rangle\, e^{i\,\mathbf{p}^T\Delta\mathbf{x}/\hbar} \tag{25.17}$$

for any operator $\hat{O}$. Here N is the number of degrees of freedom in the system, and $(\mathbf{x}_0,\mathbf{p}_0)$ is the set of initial conditions (i.e., coordinates and momenta) for a classical trajectory, $(\mathbf{x}(\mathbf{x}_0,\mathbf{p}_0),\mathbf{p}_t(\mathbf{x}_0,\mathbf{p}_0))$ being the phase point at time along this trajectory. It should also be noted that there are other approximate routes which lead to the classical Wigner model for correlation functions (other than simply postulating it). [Please see Section II-A of Ref. [543] or Section III-A of Ref. [544] for more discussion.] The LSC-IVR can be shown to be exact in the classical limit ($\hbar \to 0$), high temperature limit ($\beta \to 0$), and harmonic limit [545]. The LSC-IVR can treat both linear and nonlinear operators in a consistent way [546], can be applied to non-equilibrium as well as the above equilibrium correlation functions, and can also be used to describe electronically non-adiabatic dynamics, i.e., processes involving transitions between several potential energy surfaces. These merits of the LSC-IVR make it a versatile tool to study a variety of quantum mechanical effects in chemical dynamics of large molecular systems.

25.4.3. Local Gausssian approximation

Calculation of the Wigner function for operator $\hat{B}$ in Eq. (25.16) is usually straight-forward; in fact, is often a function only of coordinates or only of momenta, in which case its Wigner functions is simply the classical function itself. Calculating the Wigner function for operator A^{β} however, involves the Boltzmann operator with the total Hamiltonian of the complete system, so that carrying out the multidimensional Fourier transform to obtain it is far from trivial. Furthermore, it is necessary to do this in order to obtain the distribution of initial conditions of momenta for the real time trajectories. To accomplish this task, the local Gaussian approximation (LGA) proposed by Liu and Miller [543], which can be viewed as an improved version of the local harmonic approximation (LHA),

[547] and which is able to consistently treat the entire imaginary frequency regime, has been implemented in Amber. Below we briefly summarize the LGA. As in the standard normal-mode analysis, mass-weighted Hessian matrix elements are given by

$$H_{kl} = \frac{1}{\sqrt{m_k m_l}} \frac{\partial^2 V}{\partial x_k \partial x_l} \tag{25.18}$$

where represent the mass of the k-th degree of freedom. The eigenvalues of the mass-weighted Hessian matrix produce normal-mode frequencies $\{\omega_k\}$ i.e.,

$$\mathbf{T}H\mathbf{T} = \lambda \tag{25.19}$$

with λ a diagonal matrix with the elements $\left\{(\omega_k)^2\right\}$ and T an orthogonal matrix. If is the diagonal 'mass matrix' with elements $\{m_k\}$, then the mass-weighted normal mode coordinates and momenta $(\mathbf{X}_0, \mathbf{P}_0)$ are given in terms of the Cartesian variables $(\mathbf{x}_0, \mathbf{p}_0)$ by

$$\mathbf{X}_0 = \mathbf{T}^T \mathbf{M}^{1/2} \mathbf{x}_0 \tag{25.20}$$

and

$$\mathbf{P}_0 = \mathbf{T}^T \mathbf{M}^{-1/2} \mathbf{p}_0 \tag{25.21}$$

The Fourier transform of Eq. (25.17) then gives the Wigner function of as

$$\begin{aligned} A_w^\beta(\mathbf{x}_0, \mathbf{p}_0) &= (2\pi\hbar)^N \langle \mathbf{x}_0 | e^{-\beta\hat{H}} | \mathbf{x}_0 \rangle \\ &\times \prod_{k=1}^{N} \left[\left(\frac{\beta}{2\pi Q(u_k)} \right)^{1/2} \exp\left[-\beta \frac{(P_{0,k})^2}{2Q(u_k)} \right] \right] f_A(\mathbf{x}_0, \mathbf{p}_0) \end{aligned} \tag{25.22}$$

where $u_k = \beta\hbar\omega_k$, $P_{0,k}$ is the k-th component of the mass-weighted normal-mode momentum in Eq. (25.21) and the quantum correction factor is given by

$$Q(u) = \begin{cases} \dfrac{u/2}{\tanh(u/2)} & for\ real\ u \\ = \dfrac{1}{Q(u_i)} = \dfrac{\tanh(u_i/2)}{u_i/2} & for\ imaginary u \ \ (u = iu_i) \end{cases} \tag{25.23}$$

In Eq. (25.22),

$$f_A(\mathbf{x}_0, \mathbf{p}_0) = \frac{\int \mathbf{dx} \frac{\left\langle \mathbf{x}_0 - \frac{\Delta\mathbf{x}}{2} \middle| \hat{A}^\beta \middle| \mathbf{x}_0 + \frac{\Delta\mathbf{x}}{2} \right\rangle}{\langle \mathbf{x}_0 | e^{-\beta\hat{H}} | \mathbf{x}_0 \rangle} e^{i\Delta\mathbf{x}^T \cdot \mathbf{p}_0/\hbar}}{\int \mathbf{d}\Delta\mathbf{x} \frac{\left\langle \mathbf{x}_0 - \frac{\Delta\mathbf{x}}{2} \middle| e^{-\beta\hat{H}} \middle| \mathbf{x}_0 + \frac{\Delta\mathbf{x}}{2} \right\rangle}{\langle \mathbf{x}_0 | e^{-\beta\hat{H}} | \mathbf{x}_0 \rangle} e^{i\Delta\mathbf{x}^T \cdot \mathbf{p}_0/\hbar}} \tag{25.24}$$

is a function depending on the operator $\hat{A}^\beta$. For example, when $\hat{A}^\beta = e^{-\beta\hat{H}} \hat{\mathbf{x}}$, one has

$$f_A(\mathbf{x}_0, \mathbf{p}_0) = \mathbf{x}_0 + \frac{i\beta\hbar}{2} \mathbf{M}^{-1/2} \mathbf{T} \mathbf{Q}(\mathbf{u})^{-1} \mathbf{P}_0 \tag{25.25}$$

where is the diagonal quantum correction factor matrix with the elements $\{Q_k \equiv Q(u_k)\}$. Using Eq. (25.24), one can figure out the quantity $f_A(\mathbf{x}_0, \mathbf{p}_0)$ for different operators $\hat{A}^\beta$.

The explicit form of LSC-IVR correlation function (Eq. (25.16)) with the LGA is thus given by

$$\begin{aligned} C_{AB}^{LSC-IVR}(t) = \frac{1}{Z} \int \mathbf{dx}_0 \langle \mathbf{x}_0 | e^{-\beta\hat{H}} | \mathbf{x}_0 \rangle \int \mathbf{dP}_0 \prod_{k=1}^{N} \left[\left(\frac{\beta}{2\pi Q(u_k)} \right)^{1/2} \exp\left[-\beta \frac{(P_{0,k})^2}{2Q(u_k)} \right] \right] \\ \times f_A(\mathbf{x}_0, \mathbf{p}_0) B(\mathbf{x}_t, \mathbf{p}_t) \end{aligned} \tag{25.26}$$

Please refer to Refs. [543] and [544] for more detail about the LSC-IVR and the LGA.

25.4.4. Input parameters for LSC-IVR in Amber

In order to perform LSC-IVR in Amber, two additional flags should be added in the regular input script file for classical molecular dynamics.

ilscivr	Flag for LSC-IVR **=1** defines LSC-IVR
icorf_lsc	Switch for different kinds of correlation functions. It only affects the output files for LSC-IVR. **=1** defines the linear operator $\hat{A} = \hat{\mathbf{x}}$ or $\hat{A}^{\beta} = e^{-\beta\hat{H}}\hat{\mathbf{x}}$; **=2** defines the linear operator $\hat{A} = \hat{\mathbf{p}}$ or $\hat{A}^{\beta} = e^{-\beta\hat{H}}\hat{\mathbf{p}}$ **=3** defines any nonlinear operator; **=4** defines Kubo-tranformed operator $\hat{\mathbf{p}}^{\beta}_{Kubo} = \frac{1}{\beta}\int_0^{\beta} \mathrm{d}\lambda e^{-(\beta-\lambda)\hat{H}}\hat{\mathbf{p}}e^{-\lambda\hat{H}}$

Besides the above two parameters, a file *LSCrhoa.dat* is necessary. Put any integer between 0 and 10000 in the file. This parameter (*nrand*) is used for generating the initial random number for LSC-IVR. As the number of trajectories for LSC-IVR is increased, it will automatically be updated. For example, prepare the input file *lsc.in* for running a LSC-IVR trajectory inin Amber with liquid water (216 water molecules in a cell with the periodic boundary condition) with the q-SPC/fw model.

```
&cntrl
  ilscivr = 1, icorflsc = 4, ntt = 0, irest = 0, temp0 = 298.15,
  dt = 0.0005, nstlim = 2800, ntb = 1, jfastw = 4, ntpr = 1, cut = 7.0,
  ntwx = 1 ntwv = 1 ntx = 1
/
&ewald skinnb=2.0d0 /
```

As one can see, except the first two parameters for LSC-IVR, all other parameters in the input file *lsc.in* are the same as those in conventional molecular dynamics. Besides *lsc.in*, another input file *lsc.crd* for the initial coordinates of the system should be prepared (i.e., from one of the path integral bead). For instance, the command to run a parallel job with 2 processors for LSC-IVR in Amber is

```
mpirun -np 2 sander.MPI -O -i lsc.in -p watqspcfw216.top -c lsc.crd -o lsc.out
```

Here *lsc.out* is the output file to monitor the trajectory as the one for molecular dynamics. After running the above command once, the file *LSCrhoa.dat* will be updated. The first number is nrand, which will be increased by one (*nrand* = *nrand* + 1). Followed are the initial coordinates and momenta for the trajectories. If one sets *icorflsc* =4, then the function $f_A(\mathbf{x}_0, \mathbf{p}_0)$ for $\hat{\mathbf{p}}^{\beta}_{Kubo}$ will also be written into the file *LSC-rhoa.dat*. If *icorflsc* =3, then the quantum correction factor and the matrix that diagonalizes the Hesian matrix are recorded in another file LSCTmat.dat, which allows $f_A(\mathbf{x}_0, \mathbf{p}_0) B(\mathbf{x}_t, \mathbf{p}_t)$ to be evaluated for any nonlinear operators. 2) Evaluate correlation functions using LSC-IVR in Amber One may summarize the specific procedure for carrying out LSC-IVR calculation for quantum correlation functions (i.e., Eq. (13)) as follows:

1. Use path integral molecular dynamics (PIMD) in Amber to simulate the system at equilibrium. $\langle \mathbf{x}_0 | e^{-\beta\hat{H}} | \mathbf{x}_0 \rangle / Z$ is evaluated by the PIMD.

2. At specific time steps in the PIMD, randomly select one path integral bead as the initial configuration for the real time dynamics in LSC-IVR.

3. LSC-IVR in Amber diagonalizes the mass-weighted Hesian matrix of the potential surface to obtain the local normal mode frequencies and uses the LGA to give the Gaussian distribution for mass-weighted normal mode momenta
$$\prod_{k=1}^{N} \left(\beta/2\pi Q(u_k)\right)^{1/2} \exp\left[-\beta \left(P_{0,k}\right)^2 / \left(2Q(u_k)\right)\right]$$

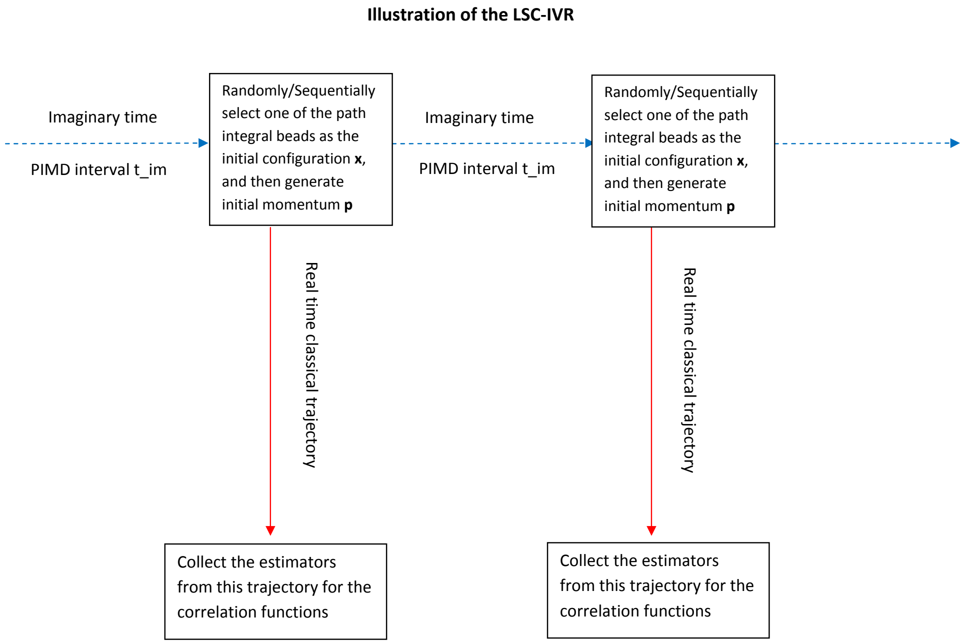


Illustration of the LSC-IVR
Imaginary time
PIMD interval t_im
Randomly/Sequentially select one of the path integral beads as the initial configuration **x**, and then generate initial momentum **p**
Imaginary time
PIMD interval t_im
Randomly/Sequentially select one of the path integral beads as the initial configuration **x**, and then generate initial momentum **p**
Real time classical trajectory
Real time classical trajectory
Collect the estimators from this trajectory for the correlation functions
Collect the estimators from this trajectory for the correlation functions

Figure 25.1.: *Flow chart for LSC-IVR.*

which is used to randomly sample initial Cartesian momentum $\mathbf{p}_0 = \mathbf{M}^{1/2}\mathbf{T}\,\mathbf{P}_0$ for the real time trajectory. Notice the parameter *nrand* in the file *LSCrhoa.dat* is used. LSC-IVR in Amber further runs a trajectory from the phase space point $(\mathbf{x}_0,\mathbf{p}_0)$. As in conventional molecular dynamics, the output files *mdcrd* and *mdvel* record the trajectory $(\mathbf{x}_t(\mathbf{x}_0,\mathbf{p}_0),\mathbf{p}_t(\mathbf{x}_0,\mathbf{p}_0))$. The parameter *nrand* is updated in the file *LSCrhoa.dat*, with the initial phase point $(\mathbf{x}_0,\mathbf{p}_0)$ the mass, and $f_A(\mathbf{x}_0,\mathbf{p}_0)$ for $\hat{\mathbf{p}}^{\beta}_{Kubo}$, etc. The file *LSCTmat.dat* may also be created for the quantum correction factor and the matrix .

4. The property

$$f_A(\mathbf{x}_0,\mathbf{p}_0)\,B(\mathbf{x}_t(\mathbf{x}_0,\mathbf{p}_0),\mathbf{p}_t(\mathbf{x}_0,\mathbf{p}_0))$$

for the corresponding time correlation function can be evaluated for the trajectory with the output files (mdcrd, mdvel, LSCrhoa.dat and LSCTmat.dat). One can write a short program in order to do so.

5. Repeat steps 2)-4) and sum the property

$$f_A(\mathbf{x}_0,\mathbf{p}_0)\,B(\mathbf{x}_t(\mathbf{x}_0,\mathbf{p}_0),\mathbf{p}_t(\mathbf{x}_0,\mathbf{p}_0))$$

for all real time classical trajectories until a converged result is obtained. It is worth noting that one can also sequentially select one path integral bead as the initial configuration in Step 2) each turn. Also in Step 4), one can simultaneously evaluate different

$$f_A(\mathbf{x}_0,\mathbf{p}_0)\,B(\mathbf{x}_t(\mathbf{x}_0,\mathbf{p}_0),\mathbf{p}_t(\mathbf{x}_0,\mathbf{p}_0))$$

for different correlation functions.

Here we give an example for the script file to get one real time trajectory in LSC-IVR for the water system already at equilibrium. (216 water molecules in a cell with the periodic boundary condition, 24 path integral beads used for the PIMD).

```
## This is to run PIMD for short time
mpirun -np 24 sander.MPI -ng 24 -groupfile gfpimd > sander.out
```

```
## Copy the configuration of one path integral bead
cd ..
cp -p PIMD/pimdbead1.rst LSC/lsc.crd
## This is to run LSC to get a real time trajectory
cd LSC
mpirun -np 2 sander.MPI -O -i lsc.in -p watqspcfw216.top -c lsc.crd -o lsc.out
## a.out is the executable file that the user writes to calculate
## for correlation functions of his/her own interest
./a.out
```

Here the file *gfpimd* is like

```
-O -i pimd.in -p watqspcfw216.top -c pimdbead1.rst -o pimdbead1.out
   -r pimdbead1.rst -pimdout pimd.out
-O -i pimd.in -p watqspcfw216.top -c pimdbead2.rst -o pimdbead2.out
   -r pimdbead2.rst -pimdout pimd.out
......
-O -i pimd.in -p watqspcfw216.top -c pimdbead24.rst -o pimdbead2.out
   -r pimdbead2.rst -pimdout pimd.out
```

For collecting many LSC-IVR trajectories, one can repeatedly use the commands in the script file with the only change for sequentially copying a path integral bead for the initial configuration for LSC-IVR, i.e.,

```
cp -p PIMD/pimdbead*.rst LSC/lsc.crd
```

Here "*" is replaced by one of the numbers 1-24 each time circularly.

All the force fields available for conventional molecular dynamics in Amber can be used for LSC-IVR. However, one should also keep in mind that some empirical force fields may require additional reparameterization (such as q-SPC/fw for SPC/fw) for approximated quantum dynamical methods such as LSC-IVR, CMD and RPMD.

25.5. Reactive Dynamics

25.5.1. Path integral quantum transition state theory

The path integral quantum transition state theory rate [548] is given by

$$k_{\mathrm{PI-QTST}} = {}^1\!/\!{}_2 \left\langle \left| \dot{\xi} \right| \right\rangle_{\xi^\ddagger} \rho_c\left(\xi^\ddagger\right) \tag{25.27}$$

where the centroid density

$$\rho_c\left(\xi\right) = \frac{\int d\mathbf{r}^{(1)} d\mathbf{r}^{(2)} \cdots d\mathbf{r}^{(P)} \exp\left[-\beta\Phi(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)})\right] \delta\left[\tilde{\xi}_c(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)}) - \xi\right]}{\int d\mathbf{r}^{(1)} d\mathbf{r}^{(2)} \cdots d\mathbf{r}^{(P)} \exp\left[-\beta\Phi(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)})\right] h\left[\xi^\ddagger - \tilde{\xi}_c(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)})\right]} \tag{25.28}$$

is related to the potential of mean force $w(\xi)$ as

$$\rho_c\left(\xi\right) = \frac{\exp\left[-\beta w\left(\xi\right)\right]}{\int_{-\infty}^{\xi^\ddagger} d\xi \exp\left[-\beta w\left(\xi\right)\right]} \tag{25.29}$$

In Eq. (25.28), $\beta = 1/k_B T$, Φ is the effective potential (see Eqs. 25.8 and 25.49), h is the Heaviside step function, $\xi^\ddagger$ is the location of the dividing surface that partitions the reactant and product regions and $\tilde{\xi}_c$ is the value of the reaction coordinate as a function of the centroid coordinates $\mathbf{r}^{(c)} = \frac{1}{P}\sum_{s=1}^{P}\mathbf{r}^{(s)}$. As Eq. (25.29) suggests, the centroid density factor can be computed using umbrella sampling approaches to generate a set of biased distributions that then can be combined into a PMF using the WHAM approach [252–254]. The dynamical frequency factor can be approximated by the velocity of a free particle along the reaction coordinate direction

$$\left\langle \left|\dot{\xi}\right| \right\rangle_{\xi^{\ddagger}} = \left(\frac{2}{\pi\beta}\right)^{1/2} \left\langle \left(\sum_{i=1}^{3N} \frac{1}{m_i} \left(\frac{\partial \tilde{\xi}_c}{\partial r_i^{(c)}} \right)^2 \right)^{1/2} \right\rangle_{\xi^{\ddagger}} \tag{25.30}$$

where $\langle \cdots \rangle_{\xi^{\ddagger}}$ denotes the conditional average computed at the dividing surface $\xi^{\ddagger}$. Both factors in the PI-QTST rate expression can be computed using the EVB/LES-PIMD facility in Amber (see Section 8.4). The value of the centroid reaction coordinate and the velocity of a free particle along the centroid RC direction are written to the file *evbout*. To output $\left|\dot{\xi}\right|$, set the variable **out_RCdot = .true.** in the EVB input file.

25.5.2. Quantum Instanton

The Quantum Instanton (QI) is a theoretical approach for computing thermal reaction rates in complex molecular systems, which is related to an older semiclassical (SC) theory of reaction rates that came to be known as the "instanton" approximation[549]. The SC instanton approximation is based on a SC approximation for the Boltzmann operator, $\exp(-\beta H)$, which involves a classical periodic orbit in pure imaginary time (or equivalently in real time on the upside-down potential energy surface) plus harmonic fluctuations about it[549]. The essential feature of the QI rate constant [550] is that it is expressed wholly in terms of the quantum Boltzmann operator, which can be evaluated for complex molecular systems using the path-integral methods described in Sec. 25.1.

In the following we will briefly illustrate the basic principles, and we will derive the fundamentals of the QI approach. We strongly recommend the user to consult the relevant literature for a more rigorous description[550, 551].

The derivation begins with the following formally exact expression of the quantum mechanical thermal rate constant[549]:

$$k(T)Q_r(T) \equiv kQ_r = \frac{1}{2\pi\hbar} \int dE e^{-\beta E} N(E), \tag{25.31}$$

where $Q_r(T)$ is the reactant partition function per unit volume at temperature T, β is the inverse temperature $1/k_B T$, and *N(E)* is the cumulative reaction probability at total energy E[537]:

$$N(E) = \frac{(2\pi\hbar)^2}{2} tr \left[\hat{F}_a \delta\left(E - \hat{H}\right) \hat{F}_b \delta\left(E - \hat{H}\right) \right]. \tag{25.32}$$

In Eq. 25.32 the flux operators $\hat{F}_a$ and $\hat{F}_b$ are defined by

$$\hat{F}_\gamma = \frac{i}{\hbar} \left[\hat{H}, h\left(\xi_\gamma(\mathbf{q})\right) \right], \tag{25.33}$$

where $\gamma = a, b$, $h\left(\xi_\gamma\right)$ is the Heaviside function, and $\hat{H}$is the Hamiltonian of the system. We note that Eqs. 25.32 and 25.33 involve two dividing surfaces $\xi_a(\mathbf{q}) = 0$ and $\xi_b(\mathbf{q}) = 0$. The microcanonical density operator $\delta\left(E - \hat{H}\right)$ in Eq. 25.32 as well as the integral over the total energy in Eq. 25.31 can be computed using a semiclassical approximation, which results in the following quantum instanton expression for the rate constant:

$$k \simeq k_{QI} \equiv \frac{1}{Q_r} C_{ff}(0) \frac{\sqrt{\pi}}{2} \frac{\hbar}{\Delta H}. \tag{25.34}$$

In Eq. 25.34, $C_{ff}(0)$ is the zero time value of the flux-flux correlation function generalized to the case of two separate dividing surfaces,

$$C_{ff}(t) = tr \left[e^{-\beta \hat{H}/2} \hat{F}_a e^{-\beta \hat{H}/2} e^{i\hat{H}t/\hbar} \hat{F}_b e^{-i\hat{H}t/\hbar} \right], \tag{25.35}$$

and ΔH is a specific type of energy variance given by

$$\Delta H^2 = \frac{tr\left[\hat{\Delta}_a e^{-\beta\hat{H}/2}\hat{H}^2\hat{\Delta}_b e^{-\beta\hat{H}/2}\right] - tr\left[\hat{\Delta}_a e^{-\beta\hat{H}/2}\hat{H}\hat{\Delta}_b e^{-\beta\hat{H}/2}\hat{H}\right]}{tr\left[\hat{\Delta}_a e^{-\beta\hat{H}/2}\hat{\Delta}_b e^{-\beta\hat{H}/2}\right]} \tag{25.36}$$

with $\hat{\Delta}_a$ and $\hat{\Delta}_b$being a modified version of the Dirac delta function:

$$\hat{\Delta}_\gamma = \Delta\left(\xi_\gamma(\hat{\mathbf{q}})\right) \equiv \delta\left(\xi_\gamma(\hat{\mathbf{q}})\right) \mid m^{-1/2}\nabla\xi_\gamma(\hat{\mathbf{q}}) \mid \tag{25.37}$$

where $\gamma = a, b$. It has been shown that ΔH can also be expressed in terms of the “delta-delta” correlation function[551].

The QI rate constant [Eq. (25.34)] can be rewritten in the form [551]

$$k_{QI} = \frac{C_{dd}(0)}{Q_r}\left\{\frac{C_{ff}(0)}{C_{dd}(0)}\frac{\sqrt{\pi}}{2}\frac{\hbar}{\triangle H}\right\} \tag{25.38}$$

where

$$\frac{C_{dd}(0;\xi_a,\xi_b)}{Q_r} = \frac{\int d\mathbf{r}^{(1)}d\mathbf{r}^{(2)}\cdots d\mathbf{r}^{(P)}\exp\left[-\beta\Phi(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)})\right]\delta\left[\tilde{\xi}(\mathbf{r}^{(P)})-\xi_a\right]\delta\left[\tilde{\xi}(\mathbf{r}^{(P/2)})-\xi_b\right]}{\int d\mathbf{r}^{(1)}d\mathbf{r}^{(2)}\cdots d\mathbf{r}^{(P)}\exp\left[-\beta\Phi(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)})\right]h\left[\xi^{\ddagger}-\tilde{\xi}(\mathbf{r}^{(P)})\right]h\left[\xi^{\ddagger}-\tilde{\xi}(\mathbf{r}^{(P/2)})\right]} \tag{25.39}$$

$$C_{ff}(0)/C_{dd}(0) = \left\langle f_v\left(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)}\right)\right\rangle_{\xi^{\ddagger}_{(P)},\xi^{\ddagger}_{(P/2)}} \tag{25.40}$$

$$\triangle H^2 = \frac{1}{2}\left\langle F\left(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)}\right)^2 + G\left(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)}\right)\right\rangle_{\xi^{\ddagger}_{(P)},\xi^{\ddagger}_{(P/2)}} \tag{25.41}$$

The conditional average $\langle\cdots\rangle_{\xi^{\ddagger}_{(P)},\xi^{\ddagger}_{(P/2)}}$ is computed from the ensemble sampled with the P and $P/2$ slices constrained to the dividing surface

$$\langle\cdots\rangle_{\xi^{\ddagger}_{(P)},\xi^{\ddagger}_{(P/2)}} = \frac{\int d\mathbf{r}^{(1)}d\mathbf{r}^{(2)}\cdots d\mathbf{r}^{(P)}\exp\left[-\beta\Phi(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)})\right]\delta\left[\tilde{\xi}(\mathbf{r}^{(P)})-\xi_a\right]\delta\left[\tilde{\xi}(\mathbf{r}^{(P/2)})-\xi_b\right]\times(\cdots)}{\int d\mathbf{r}^{(1)}d\mathbf{r}^{(2)}\cdots d\mathbf{r}^{(P)}\exp\left[-\beta\Phi(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)})\right]\delta\left[\tilde{\xi}(\mathbf{r}^{(P)})-\xi_a\right]\delta\left[\tilde{\xi}(\mathbf{r}^{(P/2)})-\xi_b\right]}$$

where the quantities within the average (for the simple case of a single quantized nuclear particle) are defined as follows:

$$f_v\left(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)}\right) = m\left(\frac{iP}{2\hbar\beta}\right)^2\nabla\xi_a\left(\mathbf{r}^{(P)}\right)\cdot\left(\mathbf{r}^{(1)}-\mathbf{r}^{(P-1)}\right)\times\nabla\xi_b\left(\mathbf{r}^{(P/2)}\right)\cdot\left(\mathbf{r}^{(P/2+1)}-\mathbf{r}^{(P/2-1)}\right) \tag{25.42}$$

$$F\left(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)}\right) = -\frac{mP}{\hbar^2\beta^2}\left\{\sum_{k=1}^{P/2}-\sum_{k=P/2+1}^{P}\right\}\left(\mathbf{r}^{(k)}-\mathbf{r}^{(k-1)}\right)^2+\frac{2}{P}\left\{\sum_{k=1}^{P/2-1}-\sum_{k=P/2+1}^{P-1}\right\}V\left(r^{(k)}\right) \tag{25.43}$$

$$G\left(\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(P)}\right) = \frac{2dP}{\beta^2}-\frac{4mP}{\hbar^2\beta^3}\sum_{k=1}^{P}\left(\mathbf{r}^{(k)}-\mathbf{r}^{(k-1)}\right)^2 \tag{25.44}$$

All factors needed to calculate the QI rate can be obtained from the EVB/LES-PIMD facility in Amber (see Section 8.4). For example, the joint distribution function [Eq. (25.39)] is computed using umbrella sampling along the reaction coordinates of the P and $P/2$ slices. The DG EVB input file of the RS malonaldehyde system may contain the following specifications:

```
&evb nevb = 2, nUFF = 1, nbias = 2, ntw_evb = 50,
dia_type = "ab_initio",
xch_type = "dist_gauss",
evb_dyn = "qi_dbonds_pmf",
dia_shift(1)%st = 1, dia_shift(1)%nrg_offset = 0.0,
dia_shift(2)%st = 2, dia_shift(2)%nrg_offset = 0.0,
dbonds_umb(1)%iatom = 8, dbonds_umb(1)%jatom = 9, dbonds_umb(1)%katom = 7,
dbonds_umb(1)%k = 100.0, dbonds_umb(1)%ezero = -.20,
dbonds_umb(2)%iatom = 8, dbonds_umb(2)%jatom = 9, dbonds_umb(2)%katom = 7,
dbonds_umb(2)%k = 100.0, dbonds_umb(2)%ezero = .40,
dist_gauss%stype = "no_dihedrals",
dist_gauss%lin_solve = "diis",
dist_gauss%xfile_type = "gaussian_fchk",
ts_xfile(1) = "malonaldehydeTS_35.fchk",
min_xfile(1) = "malonaldehydeR_35.fchk",
min_xfile(2) = "malonaldehydeP_35.fchk",
dgpt_alpha(1) = 0.72,
dgpt_alpha(2) = 0.72,
dgpt_alpha(3) = 0.72,
UFF(1)%iatom = 7, UFF(1)%jatom = 9
/
```

where the variable **evb_dyn = "qi_dbonds_pmf"** requests biased sampling along a difference of distances RC on the P and $P/2$ slices whose umbrella parameters are specified in **dbonds_umb(:)**. Input specifications for the PS malonaldehyde system is identical to the above, except that the UFF atom pair has been changed to reflect the product topology (see Section 8.5). A set of 2-dimensional (2D) biased simulations, each enhancing the sampling near a particular point of the 2D $(\xi_P \times \xi_{P/2})$ configuration space is required to map out the QI joint distribution. Using the WHAM procedure, the generated biased distributions can be unbiased to form $C_{dd}(0)/Q_r$ on the EVB ground-state surface $V_{\mathrm{el}0}$. All remaining factors involve conditional averages of f_v, F and G. These quantities are computed using umbrella sampling with the P and $P/2$ slices constrained to the dividing surface $\xi^{\ddagger} = 0.0$ and are written to the *evbout* file. The corresponding EVB input file is identical to the above, but with the following modifications:

```
⋮
~~evb_dyn      = "qi_dbonds_pmf",~~
evb_dyn      = "qi_dbonds_dyn",
⋮
~~dbonds_umb(1)%k = 100.0, dbonds_umb(1)%k = -.20,~~
dbonds_umb(1)%k = 400.0, dbonds_umb(1)%ezero = 0.0,
⋮
~~dbonds_umb(2)%k = 100.0, dbonds_umb(2)%ezero = .40,~~
dbonds_umb(2)%k = 400.0, dbonds_umb(2)%ezero = 0.0,
⋮
```

25.6. Isotope effects

25.6.1. Thermodynamic integration with respect to mass

As mentioned in Section 21.1, the standard implementation of thermodynamic integration in Amber assumes that the potential energy surface (PES) changes, but masses do not. For isotope effects the situation is exactly opposite: Within the Born-Oppenheimer approximation, the PES remains unchanged and it is the masses that change. One is usually interested in the ratio of the partition functions of the system with the heavy isotope ($Q^{(h)}$) and the light isotope ($Q^{(l)}$),

$$Q^{(h)}/Q^{(l)} = e^{-\beta\Delta F},$$

where the change in free energy ΔF can be computed by the thermodynamic integration (TI) with respect to mass as

$$\Delta F = \int_0^1 \langle dV_{\text{eff}}(\lambda)/d\lambda\rangle\, d\lambda. \tag{25.45}$$

The parameter λ interpolates between the masses of the system with the lighter ($\lambda = 0$) and the heavier ($\lambda = 1$) isotopes,

$$m_i(\lambda) = (1-\lambda)\, m_i^{(l)} + \lambda m_i^{(h)}, \tag{25.46}$$

and the effective potential V_{eff} is defined as

$$V_{\text{eff}}(\lambda) := -\beta^{-1}\log Q(\lambda). \tag{25.47}$$

The TI consists in running several simulations for different values of λ, computing $\langle dV_{\text{eff}}(\lambda)/d\lambda\rangle$ in each simulation, and performing the simple integral (Eq. 25.45) in the end.

In classical mechanics, the TI w.r.t. mass would be rather trivial, so we assume that the calculation is quantum-mechanical and uses PIMD. Let N be the number of atoms and P the number of imaginary time slices in the discretized path integral (PI). ($P = 1$ gives classical mechanics, $P \to \infty$ gives quantum mechanics.) The PI representation of Q is

$$Q \simeq \left(\frac{P}{2\pi\hbar^2\beta}\right)^{3NP/2} \prod_{i=1}^{N} m_i^{3P/2} \int d\mathbf{r}^{(1)} \cdots \int d\mathbf{r}^{(P)} e^{-\beta\Phi}, \tag{25.48}$$

where Φ is given by

$$\Phi = \frac{P}{2\hbar^2\beta^2}\sum_{i=1}^{N} m_i \sum_{s=1}^{P}\left(\mathbf{r}_i^{(s)} - \mathbf{r}_i^{(s+1)}\right)^2 + \frac{1}{P}\sum_{s=1}^{P} V\left(\mathbf{r}^{(s)}\right) \tag{25.49}$$

and $\mathbf{r}_i^{(s)}$ denotes the sth slice coordinates of the ith atom.

The tricky part in PI simulations is finding efficient ways to estimate relevant quantities (in the PI jargon, finding efficient "estimators") – in our case $dV_{\text{eff}}(\lambda)/d\lambda$ from Eq. (25.45). For example, direct differentiation of Eq. (25.48) gives the thermodynamic-like estimator (TE),[552]

$$\frac{dV_{\text{eff}}(\lambda)}{d\lambda} \simeq -\sum_{i=1}^{N} \frac{dm_i}{d\lambda}\left[\frac{3P}{2m_i\beta} - \frac{P}{2\hbar^2\beta^2}\sum_{s=1}^{P}\left(\mathbf{r}_i^{(s)} - \mathbf{r}_i^{(s+1)}\right)^2\right] \quad \text{(TE)}. \tag{25.50}$$

The problem with this estimator is that its statistical error grows with P. If one wishes to go to the quantum limit, one must increase the number of samples enormously. In Ref. [553], this drawback was avoided by subtracting the centroid coordinate

$$\mathbf{r}_i^{(C)} = \frac{1}{P}\sum_{s=0}^{P-1}\mathbf{r}_i^{(s)}$$

and using mass-scaled coordinates in Eq. (25.48). The resulting virial-like estimator (VE),

$$\frac{dV_{\text{eff}}(\lambda)}{d\lambda} \simeq -\sum_{i=1}^{N}\frac{dm_i/d\lambda}{m_i}\left[\frac{3}{2\beta} + \frac{1}{2P}\left\langle \sum_{s=1}^{P}\left(\mathbf{r}_i^{(s)} - \mathbf{r}_i^{(C)}\right)\cdot\frac{\partial V\left(\mathbf{r}^{(s)}\right)}{\partial \mathbf{r}_i^{(s)}}\right\rangle\right] \quad \text{(VE)}, \tag{25.51}$$

has the advantage that the statistical error is independent of P. Compared to TE, the virial-like estimator requires the gradient of the potential, but at no additional cost, since the gradient is already needed for the PIMD. Both types of estimators are implemented in Amber in order to provide an independent comparison, but in general the virial estimator is preferred.

Strictly speaking, the preceding derivation was for a system bound in an external potential. In molecular systems with internal interactions only, the partition function can only be defined per unit volume because the center-of-mass coordinate is unbound. However, if the sampling is done in Cartesian coordinates as in Amber, the preceding estimators remain unchanged. This can be justified by considering a finite volume V and taking a limit $V \to \infty$.

25.6.2. Amber implementation

The thermodynamic integration w.r.t. mass is run in Amber as any other PIMD simulation with the following changes.

1. In the *mdin* file, ITIMASS and CLAMBDA must be set.

   ITIMASS = 0 No thermodynamic integration w.r.t. mass (default).

   ITIMASS = 1 Run TI w.r.t. mass using the efficient virial estimator (25.51). This is the preferred value.

   ITIMASS = 2 Run TI w.r.t. mass using the simple thermodynamic estimator (25.50). This option should only be used for testing. The virial estimator (option 1) has much smaller statistical error.

   CLAMBDA Contains the value of λ for TI ($0.0 \leq \lambda \leq 1.0$) from Eq. (25.46).

2. In the *prmtop* (topology) file, a flag TI_MASS with the perturbed masses must be added. In other words, the current flag MASS includes the masses for the first (unperturbed) isotopic system ($m_i^{(l)}$), and TI_MASS includes the masses for the second (perturbed) isotopic system ($m_i^{(h)}$). Note that unlike the standard TI for which the force field changes, the TI w.r.t. mass requires only one topology file.

3. The output $dV_{\text{eff}}/d\lambda$ from Eqs. (25.50) and (25.51) for the TI is printed as "DV/DL" in the *mdout* file (as for the standard TI).

Note: Currently, the TI w.r.t. mass can be used with both implementations of the PIMD (that is the full PIMD and the LES PIMD). There are examples of both in the directory test/ti_mass.

25.6.3. Equilibrium isotope effects

Equilibrium (or thermodynamic) isotope effect (EIE) is the effect of isotopic substitution on the equilibrium constant K of a chemical reaction. Denoting the quantities pertaining to the reaction with the lighter (heavier) isotope by a superscript l (h), the EIE is defined as the ratio of the equilibrium constants

$$\text{EIE} := \frac{K^{(l)}}{K^{(h)}}. \tag{25.52}$$

Within the Born-Oppenheimer approximation, the potential energy surfaces of isotopic molecules are identical, and so the EIE is only due to the effect of the isotopic mass on the nuclear motion of the reactants and products. The EIE can be expressed as the ratio

$$\text{EIE} = \frac{Q_p^{(l)}/Q_p^{(h)}}{Q_r^{(l)}/Q_r^{(h)}}. \tag{25.53}$$

where Q_r and Q_p denote the reactant and product partition functions, respectively. Equation (25.52) suggests that the EIE can be found in practice by performing two thermodynamic integrations: for the reactants, $Q_r^{(l)}/Q_r^{(h)}$, and for the products, $Q_p^{(l)}/Q_p^{(h)}$.

25.6.4. Kinetic isotope effects

Similarly, the kinetic isotope effect (KIE) is the effect of isotopic substitution on the rate constant k of a chemical reaction, and is defined as

$$\text{KIE} := \frac{k^{(l)}}{k^{(h)}}.$$

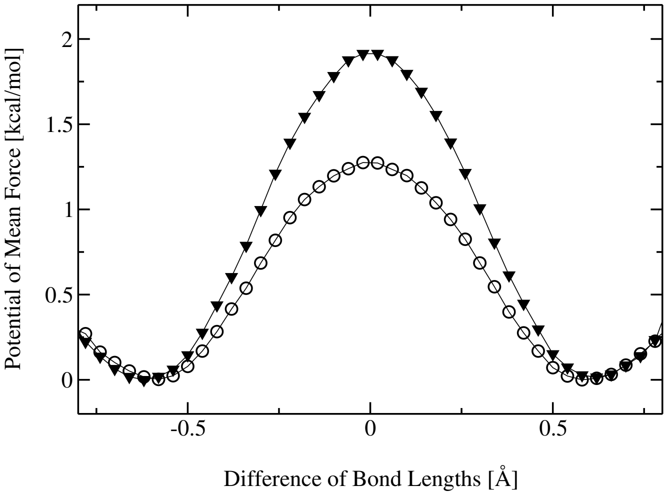


Figure 25.2.: *PMFs for proton (⊙ curve) and deuterium (▼ curve) transfer in malonaldehyde using DG EVB/LES-PIMD.*

The exact quantum-mechanical expression for the rate constant is

$$k = Q_r^{-1} tr\left(e^{-\beta\hat{H}}\hat{F}\hat{P}\right)$$

where $\hat{H}$ is the Hamiltonian operator and $\hat{F}\hat{P}$ is the reactive flux operator. Unfortunately, the exact k cannot be computed even for fairly small molecules. There exists, however, a very accurate Quantum Instanton (QI) approximation for the rate constant [550], given by

$$k_{\mathrm{QI}} = \frac{1}{2}\sqrt{\pi}\hbar\frac{C_{\mathrm{ff}}(0)}{Q_r\Delta H}. \tag{25.54}$$

In this expression, $C_{ff}(t)$ is the flux-flux correlation function and ΔH is a specific type of energy variance, defined in Ref. [550]. A path-integral implementation of the QI approximation to compute KIEs has been developed in Refs. [552] and [553]. Within this approximation, the KIE is written as a product of several factors,

$$\mathrm{KIE}_{\mathrm{QI}} = \frac{k_{\mathrm{QI}}^{(l)}}{k_{\mathrm{QI}}^{(h)}} = \frac{Q_r^{(l)}}{Q_r^{(h)}} \times \frac{\Delta H^{(h)}}{\Delta H^{(l)}} \times \frac{C_{\mathrm{dd}}^{(l)}(0)}{C_{\mathrm{dd}}^{(h)}(0)} \times \frac{C_{\mathrm{ff}}^{(l)}(0)/C_{\mathrm{dd}}^{(l)}(0)}{C_{\mathrm{ff}}^{(h)}(0)/C_{\mathrm{dd}}^{(h)}(0)}, \tag{25.55}$$

where for convenience we have multiplied and divided by so-called delta-delta correlation function $C_{\mathrm{dd}}(t)$. Using the PIMD implementation in Amber, quantities such as $\Delta H^{(h)}$ or $C_{\mathrm{ff}}^{(l)}(0)/C_{\mathrm{dd}}^{(l)}(0)$, can be computed directly in a constrained PIMD simulation because they are thermodynamic averages (see Section 25.5.2 on the QI evaluation of the rate constant). The ratio $Q_r^{(l)}/Q_r^{(h)}$ must be computed by the TI with respect to mass. Finally, the correlation function $C_{\mathrm{dd}}(t)$ is defined very similarly to the partition function Q, with the exception that it is constrained to two dividing surfaces for the reaction. Consequently, the ratio $C_{\mathrm{dd}}^{(l)}(0)/C_{\mathrm{dd}}^{(h)}(0)$ must be computed by a TI but with a constrained PIMD.

25.6.5. Estimating the kinetic isotope effect using EVB/LES-PIMD

The kinetic isotope effect is defined as the ratio of the rate of reaction involving the lighter isotope compared to the rate involving the heavier isotope, KIE $= k^{(l)}/k^{(h)}$. Within the PI-QTST approximation, the KIE involves the ratio of dynamical frequency factors and the ratio of centroid densities [see Eqs. (25.27-25.30)]. Each frequency

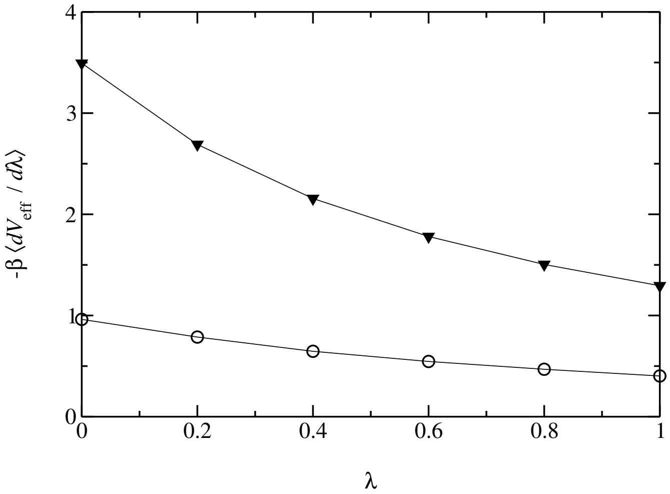


Figure 25.3.: *Average value of* $-\beta\,\langle dV_{\text{eff}}/d\lambda\rangle$ *sampled in the RS region (▼ curve) and at the dividing surface* $\xi^{\ddagger}$ *(⊙ curve) as a parameter of* λ*.*

factor can be computed using biased sampling EVB/LES-PIMD, where the umbrella potential constrains the sampling along the dividing surface. The ratio of the centroid densities can be computed by employing biased sampling (see Sections 8.3 and 8.4) to map out the PMFs for both isotope reactions or by thermodynamic integration with respect to mass (see Sections 25.6.1 and 25.6.2). The former case involves two separate PMF calculations where the respective isotope masses are specified in the `%FLAG MASS` section of the parmtop files. Figure 25.2 compares the PMFs for the isotopic substitution of the transferring proton to a deuterium. All simulation parameters used in generating the PMFs are identical, with the exception that the transferring proton mass was changed from 1.008 amu to 2.014 amu in the deuterium parmtop file. The ratio of the centroid densities using Eq. (25.29) provide a value of 2.52.

Thermodynamic integration with respect to mass is discussed in detail in Sections 25.6.1 and 25.6.2. The key quantities we need to estimate the ratio of the centroid densities are embodied in the equation [553, 554]

$$\frac{\rho_c^{(l)}\left(\xi^{\ddagger}\right)}{\rho_c^{(h)}\left(\xi^{\ddagger}\right)} = \exp\left[-\beta\int_0^1 d\lambda\left(\left\langle\frac{dV_{\text{eff}}(\lambda)}{d\lambda}\right\rangle_{\text{RS}} - \left\langle\frac{dV_{\text{eff}}(\lambda)}{d\lambda}\right\rangle_{\xi^{\ddagger}}\right)\right] \tag{25.56}$$

Thus two separate TI by mass simulations are required, one that samples $dV_{\text{eff}}/d\lambda$ in the reactant region and the other which samples along the dividing surface $\xi^{\ddagger}$. The former case requires ground-state EVB/LES-PIMD dynamics (i.e., **evb_dyn="groundstate"**) on the reactant surface with TI by mass invoked in the *mdin* file (i.e., **ievb=1, ipimd=2, ntt=4, nchain=4, itimass=1, clambda=0.2**). A set of simulations with **clambda** ranging from 0 to 1 maps out the derivative along the mass transformation progress variable. Sampling of $dV_{\text{eff}}/d\lambda$ along the dividing surface is invoked in a similar fashion, but with ground-state dynamics replaced by biased sampling constrained to the dividing surface (i.e., **evb_dyn = "dbonds_umb", dbonds_umb(1)%iatom = 8, dbonds_umb(1)%jatom = 9, dbonds_umb(1)%katom = 7, dbonds_umb(1)%k = 400.000, dbonds_umb(1)%ezero = 0.0**). Here, the dividing surface $\xi^{\ddagger}$ is located at 0.0 along the difference of distances RC. Figure 25.3 shows the averages $-\beta\,\langle\cdots\rangle_{\text{RS}}$ and $-\beta\,\langle\cdots\rangle_{\xi^{\ddagger}}$ as a parameter of λ for the malonaldehyde system. Using the integration of Eq. (25.56) provides a centroid density ratio of 2.86.

26. mdgx

David S. Cerutti

The *mdgx* simulations package is a molecular dynamics engine with functionality that mimics some of *sander* and *pmemd*, but featuring simple C code and an atom sorting routine that simplifies the flow of information during force calculations. With the availability of *pmemd* and its GPU-compatible variant for efficient, long-timescale simulations, and the extensive development of thermodynamic integration, free energy calculations, and enhanced sampling methods that has taken place in *sander*, the principal purpose of *mdgx* is to provide a tool for radical redesign of the basic molecular dynamics algorithms and models. Currently, *mdgx* supports modest parallel capabilities, but the limiting factor is load-balancing; the molecular dynamics routines are designed for much higher parallelism. The first application of *mdgx* was to demonstrate the feasibility of multiple reciprocal space meshes spanning different regions of the simulation cell at different resolutions.[555] Future applications, discussed in more detail later in this chapter, pertain to new charge distributions with significant numbers of off-atom "virtual" force centers.

While it is capable of performing molecular dynamics based on standard **prmtop** topology, **inpcrd** starting coordinates files, and input files in a format very much like the **mdin** files, it should be emphasized that the dynamcis mode of *mdgx* is really a program for experts with knowledge of the algorithms. There is currently no minimization algorithm in place, so *mdgx* cannot yet be used as a standalone program for converting coordinates from an experiment into a trajectory. However, *mdgx* does have the capability to perform dynamics in isothermal as well as isobaric ensembles while incorporating some of the more advanced features of *sander* and *pmemd*. With continued development, it is on a course to become a production molecular dynamics code for general use.

26.1. Input and Output

Input command files for *mdgx* may be similar to the **mdin** format used by *sander* and *pmemd*. One requirement of *mdgx* that is not found in *sander* is that each of the &namelist segments of the input file must begin with the identifier of the &namelist on its own line and end with the keyword &end on its own separate line. However, the &namelist format is not strictly enforced in *mdgx*, not all *sander* input variables are available in *mdgx*, and some new input variables have been added. All *mdgx* input variables can also be identified by aliases that may be lengthier than their *sander* counterparts but may make the input easier for a human to parse.

All *mdgx* &namelists and their associated variables may be browsed by running the *mdgx* program itself; running the program with no command line arguments will produce basic instructions for usage and a list of command-line arguments to display each &namelist. Certain directives to *mdgx* may be supplied as either part of the input file or on the command line; in particular, the names of the topology, input coordinates, and output files may be specified in either manner. Also, the random number generator seed and thermodynamic integration coupling parameter λ may be specified on the command line. However, if the same variable is declared both on the command line and in the input file, the command-line input will take precedence. This predominance makes it possible to execute multiple related *mdgx* runs based on a single input file. Units of input variables follow the *sander* and *pmemd* conventions.

The *mdgx* program will read standard AMBER **prmtop** files using its own routines and perform basic tests of the topology to identify common problems such as omitted disulfide bonds or "D" to "L" chirality flips in the standard amino acids; any potential problems are reported in the **mdout** output diagnostics files, but do not immediately lead the program to halt. In addition to the standard information contained in an AMBER topology file, *mdgx* is being developed to also be able to read other sorts of information given certain directives in the input command (**mdin**) file. As will be discussed later, *mdgx* is able to read auxiliary information that modifies the topology specified by a **prmtop** file, adding virtual sites or changing the nonbonded parameters of specified atoms. (These changes are not written back into the original **prmtop**.)

Output files produced by *mdgx* follow the AMBER .crd and NetCDF formats for coordinates and velocities. Forces on all atoms can also be printed over the course of a trajectory. Separate suffixes may be applied to the **mdout** output diagnostic information, trajectory files, energy, and restart files, as specified by the user. *mdgx* also has the capability to print outputs from a single trajectory into multiple segments if the user specifies a value of the *nfistep* variable that is a factor of the *sander*-related *nstlim* variable. In such a case, *mdgx* will print files of the format [base name]###.[suffix], where [base name] is a base file name supplied by the user, ### is a number of the segment beginning at zero, and [suffix] is a file extension supplied by the user (for instance, "rst" for restart files, "out" for **mdout** output diagnostics). The number of segments is determined by the ratio of *nstlim* to *nfistep*; the former indicates the total number of dynamics steps, the latter the number of steps in each segment. At the start of dynamics *mdgx* will check for the existence of complete output diagnostics and restart files (as indicated by a special three-line ASCII mark) starting at segment 0 and continuing until a missing output diagnostics file is encountered, even if file overwriting (the *sander*-related "-O" command-line option) has been specified. (Not allowing file overwriting will only cause the program to abort if, on a subsequent segment for which complete output diagnostics and restart files do not exist, some other output such as a trajectory coordinates file does exist.) The intention of this elaborate scheme is to permit one long run to be broken into many segments without halting the program, and to provide an internal means of checkpointing a run if the program must be restarted. Because the changes to the output format are potentially dramatic, the *nfistep* variable must be set deliberately; any value that is not a factor of *nstlim* will result in *nfistep* being set to zero and outputs will be printed to files named [base name].[suffix].

The *mdgx* program also provides its own output format for force diagnostics. In *sander*, information relating the bond, angle, torsion, and nonbonded direct and reciprocal space forces is only available by running in "debugging" mode as specified by the &debugf namelist block. In *mdgx*, such output is available by setting the *sander*-related *imin* variable to 2; the output is produced in ASCII format with numerous comments to make the results comprehensible to a human.

26.2. Installation

mdgx is installed as part of the AmberTools package. The program relies on the FFTW 3.3 and NetCDF libraries already distributed as part of AmberTools.

26.3. Special Algorithmic Features of mdgx

While it does not currently support the breadth of molecular dynamics algorithms offered by the *pmemd* or *sander* programs, *mdgx* does have capabilities that set it apart from other simulators in the AMBER software package.

First, *mdgx* can perform molecular simulations at constant volume with the "Multi-Level Ewald" implementation of Particle Mesh Ewald [555] electrostatics. This algorithm breaks the one reciprocal space mesh used in most Particle:Particle / Particle:Mesh techniques into multiple slabs spanning subdomains of the simulations cell and a much coarser variant of the global mesh for reuniting the subdomains. The intention of this algorithm is to provide a means for distributing and mitigating the communications required for solving the system's long-ranged electrostatics. *mdgx* also provides an implementation of standard Smooth Particle Mesh Ewald electrostatics [366], with the added generality of independent interpolation orders in each of the three mesh dimensions. These features may be accessed through control variables in the &ewald namelist.

Another feature of mdgx, ported to *pmemd* in Amber13, is the Monte-Carlo barostat, available by specifying $ntp > 0$ and barostat = 2. This remarkably simple barostat makes volume moves, rescales system coordinates to match the new unit cell dimensions, and uses the Metropolis criterion to compare the energies of the original and trial configurations:

$$\chi_{\text{acc}} = \min\left[\ 1,\quad \left(\frac{V^{\text{new}}}{V^{\text{old}}}\right)^{N} \exp\left(-\frac{1}{kT}\left(U^{\text{new}} - U^{\text{old}} + P\left(V^{\text{new}} - V^{\text{old}}\right)\right)\right)\ \right]$$

In the above formula, the probability of accepting a move χ is determined by the product of two factors. The first factor is the ratio of volumes V in the trial (new) and initial (old) configurations taken to the power of the number of particles in the system *N*. (Note that in the presence of rigid constraints, each rigidly constrained group of atoms counts as only one particle.) The second factor is a Boltzmann-weighted probability based on the sum of the potential energy of the system *U* and the pressure-volume work that the system does on its surroundings. In the above formula, the pressure *P* and temperature *T* are arbitrary parameters of the barostat: specifically, *P* is the external pressure (*pres0* in *sander* input), and *T* is set to match the external temperature of the thermostat in use. In this barostat, the system kinetic energy does not directly play a role in determining the system volume. However, in a condensed system of real particles the kinetic and potential energies quickly exchange, and even in the case of an ideal gas the two factors balance out such that the familiar ideal gas law is recovered so long as the temperature *T* given to the barostat and the actual temperature of the gas particles match. Currently, this barostat is set up to rescale the volume isotropically, but in principle anisotropic volume changes and even alterations of the unit cell angles are feasible.

A principal advantage of the Monte-Carlo barostat is that no computation of the virial is necessary: the barostat can be applied at whatever frequency the user requires, and will maintain the proper system volume so long as moves are accepted at a frequency much greater than the rate at which the system might spontaneously move to configurations which change its equilibrium volume under the applied external pressure *P*. The default of attempting one barostat move by increasing or decreasing the system volume by up to one-tenth of a percent of its initial volume appears to result in a good acceptance ratio. The default of attempting the moves every 100 steps of dynamics, about every 100 to 200 femtoseconds, should be sufficient to accommodate most processes of interest and produce sound equilibrium statistics on the timescale of nanoseconds. Because the moves only require recalculation of the energy (which is done with merely a few additions and multiplications in special cases), the Monte-Carlo barostat may also have a speed advantage over the methods currently implemented in *sander* and *pmemd*.

26.4. Customizable Virtual Site Support in mdgx

It is not completely feasible to perform molecular dynamics with massless particles. However, for many useful cases in which the locations of massless particles are determined by the locations of two or more atoms with mass, it is possible to perform dynamics by using the chain rule to transfer forces from the “virtual sites” to the massive particles. These constructions, enumerated below, provide a means for breaking out of the “one atom, one site” paradigm that has dominated classical molecular dynamics, but the **prmtop** format utilized by the *sander* and *pmemd* programs does not always provide a straightforward means of expressing the relationships between virtual sites and their parent (or “frame”) atoms and the sander and pmemd programs only support the most widely used cases of virtual sites (e.g. TIP4P and TIP5P water).

The *mdgx* program provides a means for adding any number of virtual sites to an existing force field, with custom charges and even Lennard-Jones properties. The only limitations with the virtual sites are that no new bonded terms may be added, that the virtual sites carry zero mass, and that each virtual site location be determined by two or three frame atoms on the same residue which do have mass. The constructions below follow those outlined in the GROMACS manual; a four-point frame construction devised by the GROMACS team is not yet implemented, but a “zeroth” frame type is available in mdgx which allows, without changing the **prmtop**, run-time modification of existing atomic non-bonded parameters.

In the Fig. 26.1, the &rule namelist variables for specifying each virtual site constructor are superimposed on atoms, vectors, and angles. In Style 1, the virtual site lies along the line determined by two atoms; v12 denotes the fraction of the distance between the two atoms at which to place the virtual site. In Style 2, the virtual site lies in the plane determined by three atoms at a point determined by a combination of the displacements between atoms 1 and 2 and atoms 2 and 3. Virtual sites of Styles 1 and 2 are located by linear combinations of the positions of their frame atoms. In Style 3, the virtual site is located along the line described by frame atom 1 and a point between frame atoms 2 and 3 (v23 denoting the fraction of this distance), at a fixed distance v1e from frame atom 1. Style 4, perhaps the most mathematically challenging frame type to define but very useful and intuitively comprehensible, places a virtual site at a fixed distance v1e from frame atom 1 such that the angle illustrated has the value theta (specified in radians in the &rule namelist). The virtual site remains in the plane of the frame atoms, and frame atom 3, which must not be colinear with the other frame atoms, orients the sign of theta. Virtual sites of Style 5 are defined as sites of Style 2, but projected normal to the plane according to a multiple v12x13 of the cross product

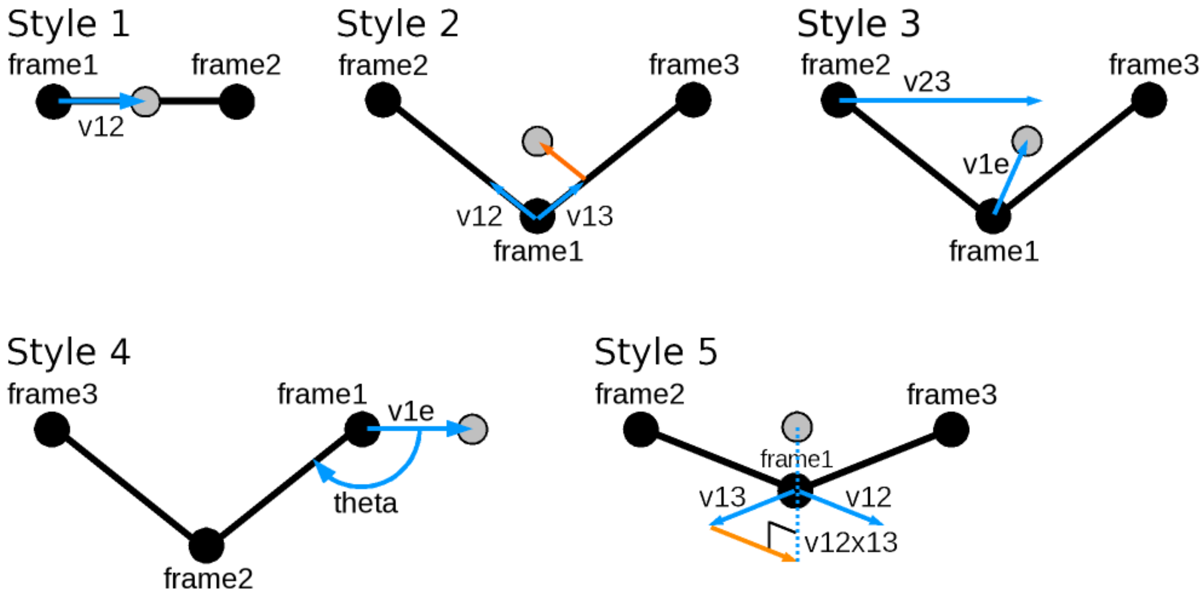


Figure 26.1.: *Frame styles in mdgx.*

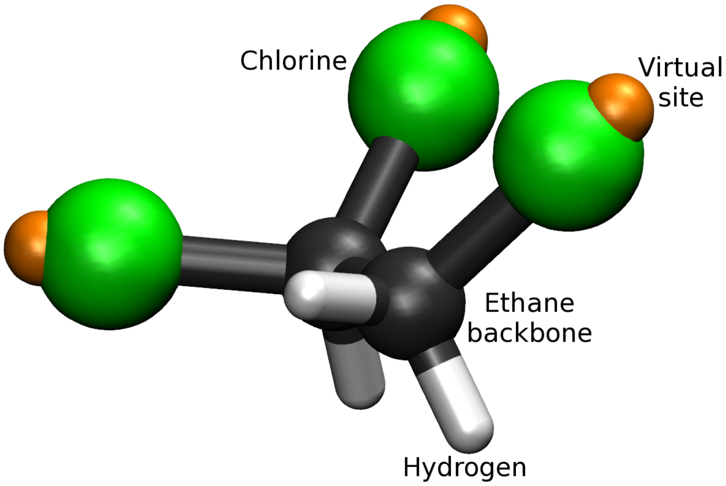
of the vectors between frame atoms 1 and 2 and frame atoms 1 and 3. Note that virtual sites of Styles 1, 2, and 5 will stretch with their frames, whereas 3 and 4 will not. The stretching will be minor if the frame atoms are bonded as shown in the figure. Due to the manner in which virtual sites are positioned in *mdgx*, frame atoms 2 and 3, and the virtual site when placed, must lie within half the van-der Waals non-bonded cutoff of frame atom 1. This should seldom if ever be a problem. A complete list of &rule namelist variables follows (shorthand aliases in parentheses):

- **FrameAtom**[1,2,3] (frame[1,2,3]): specifies the frame atoms needed for virtual site construction
- **ExtraPoint** (epname): The name of the virtual site
- **AtomName** (atom): The name of the virtual site (alternate specifications)
- **FrameStyle** (style): The frame style to use (see descriptions in the preceding figure); acceptable values are 0 through 5
- **Exclude**[2,3] (excl[2,3]): The virtual site is definitively 1:1 bound to frame atom 1 and thereby inherits all 1:2, 1:3, and 1:4 neighbors of frame atom 1, but if ? is 2 or 3 then the virtual site will also be considered 1:1 to frame atoms 2 or 3 and inherit their bonded neighbors as well. This will not affect the 1:2, 1:3, and 1:4 neighbor lists of the frame atoms themselves.
- **Vector12** (v12): Defined according to frame type; see preceding paragraph and illustration.
- **Vector1E** (v1e): Defined according to frame type; see preceding paragraph and illustration.
- **Vector13** (v13): Defined according to frame type; see preceding paragraph and illustration.
- **Theta** (theta): Defined according to frame type; see preceding paragraph and illustration.
- **Vector23** (v23): Defined according to frame type; see preceding paragraph and illustration.
- **Vector12x13** (v12x13): Defined according to frame type; see preceding paragraph and illustration.
- **Charge** (q): Charge of the virtual site

- **Sigma** (sig): Lennard-Jonesσ parameter for the virtual site
- **Epsilon** (eps): Lennard-Jones ε parameter for the virtual site
- **ResidueName** (residue): The residue to which extra points will be added. Because it is specified according to the four-character name, there is some possibility for ambiguity as terminal residues often have the same names as residues in the middle of a chain. Therefore, in order to add a virtual site to an the amino terminus of N-terminal alanine but skip over alanines within a polypeptide, the N-terminal alanine would have to be given a new name within the **prmtop**.

The purpose of the zeroth frame type is to round out a temporary solution to the problem of testing virtual sites configurations in Amber; ultimately, the best solution is to incorporate all virtual site constructions into LEaP and expand the **prmtop** format to accommodate them. However, for experimentation and validation the *mdgx* approach of adding particles to an existing topology is straightforward, faster than creating new topologies starting with antechamber, and will remain available as part of the program for the foreseeable future. It is possible in mdgx (noting that the rigid geometry of the massive atoms is the same throughout all TIP water models) to simulate TIP4P[74] or TIP5P[74] water starting from a **prmtop** containing TIP3P water, although it is more convenient and perhaps marginally faster to simulate beginning with a **prmtop** specifying the more complex water model.

Virtual sites added in this manner follow the neighbor conventions described in Subsection 17.7.4: virtual sites are counted as “1:1” neighbors of their first parent atoms and then inherit all 1:2 (bond), 1:3 (angle), and 1:4 nonbonded neighbors of the first parent atom. It is also possible to endow virtual sites with neighbors of other parent atoms, effectively declaring the virtual sites to be 1:1 neighbors of more than one atom. The neighbor list updates implied by adding virtual sites do not get applied retroactively, however, so multiple frame atoms do not become 1:1 neighbors of each other. Because of the exclusions implied by different frame constructions, care should be taken when defining parent atoms. For instance, in the chlorinated ethane derivative below virtual sites of frame type 1 (v12 = –0.3, with chlorines being frame atom 1 and the bonded carbons being frame atom 2) can be shown to significantly improve the electrostatic fit to quantum-mechanical MP2 calculations.



In principle, the frame atom 1 may be defined as the carbon, with the chlorine (which is actually closest to the virtual site) merely defining the direction of the virtual site projection. However, this construction omits interactions between virtual sites on opposite ends of the molecule, and as a result the torsional conformations of the molecule are drastically altered (so much so that the hydration free energy in explicit solvent simulations changes by more than 3 kcal/mol). If the chlorines themselves are made frame atom 1 in each virtual site frame, the virtual sites become 1:4 neighbors to one another and interact by a slightly screened electrostatic potential. The effects on the torsional distribution and resulting hydration free energy are then much more modest. This trichloroethane represents an extreme case, but more subtle examples abound. In general, virtual sites can change the charge distribution of a molecule to roughly the same degree that refitting an atom-centered charge model to new quantum data does. Ideally, torsional parameters would be refitted in all cases to accommodate the new electrostatic model.

26.5. Implicitly Polarized Charge Development in mdgx

The *mdgx* package is the workhorse used to create ff14ipq and its successor ff15ipq. At the heart of each of these protein force fields is the Implicitly Polarized Charge model (IPolQ), which *mdgx* automates. The purpose of IPolQ is to derive an appropriate set of fixed partial charges on a molecule which account for the mean-field polarization it displays in solvent (water) while also accounting for the energetic cost of perturbing the gas phase wave function. This is handled by two separate quantum calculations on the same set of molecular coordinates: the first is performed in vacuum, the second in the presence of a time-averaged solvent charge density. Taking a page from linear response theory, the average of the two calculations provides the target electrostatic potential that the fixed partial charges should project. Collecting that charge density, particularly if the solvent contains counterions and there are infinite electrostatics in play, takes a fair amount of code, but this is why *mdgx* has a special module. It currently interfaces with two quantum packages, ORCA and Gaussian, to drive the QM calculations, and additional code in this module will write inputs and post-process the outputs of each program into material suitable for the *mdgx* electrostatic potential fitting tools.

The variables available in the &ipolq namelist are as follows (shorthand aliases in parentheses):

- **SoluteMol** (solute): The solute molecule, specified by an ambmask string. This is the molecule of interest for charge fitting, and will be immobilized during the simulation. This must be specified by the user.
- **FrameRate** (ntqs): The rate of charge density sampling; the number of steps between successive snapshots to determine the solvent reaction field potential (SRFP). Default 1000 (the time step set in the &cntrl or &ipolq namelists should factor into the ntqs setting).
- **FrameCount** (nqframe): The number of frames used to compose the SRFP. Default 10 (this is too low for most solvent environments).
- **EqStepCount** (nsteqlim): The number of steps used to equilibrate the system before charge density collection begins. Use this part of the simulation to buffer against any artifacts that might arise from suddenly freezing the solute in place. Default 10000.
- **Blocks** (nblock): The number of blocks into which the simulation shall be divided for the purpose of estimating the convergence of the electrostatic potential. Default 4.
- **Verbose** (verbose): Default 0; set to 1 to activate step-by-step progress updates printed to the terminal window. Useful for monitoring short runs to ensure that the input successfully completes the SRFP calculation.
- **EConverge** (econv): Convergence tolerance for the SRFP (convergence checking is not yet implemented).
- **QShellCount** (nqshell): The number of additional shells of charge to place around the system in order to approximate the SRFP due to infinite electrostatics in the confines of an isolated system. Maximum (and default) is 3, minimum is 1.
- **VShellCount** (nvshell): The number of shells around each atom on which the exact SRFP due to infinite electrostatics shall be calculated. Maximum (and default) is 3, minimum is 1.
- **QSpherePts** (nqphpt): In order to generate the surface charges that will help in approximating the SRFP, this number of points is placed equidistant on a sphere. The sphere is then rotated randomly and expanded to the radii indicated by qshell[1,2,3,x]. All points that are on the sphere due to one atom but within the sphere projected by another atom are deleted, until only points on the proper surface remain. Default 100.
- **VSpherePts** (nvphpt): Similar to nqphpt above, but for the shells of SRFP evaluation points. Default 20.
- **ExpQBoundary** (qshell1): The distance at which to locate the first surface charges, and to stop collecting charges explicitly from the simulation's non-solute (that is, solvent) atoms. Default 5.0.
- **QShell[2,3]** (qshell2, qshell3): The distance at which to locate the second and third shells of boundary charges. If engaged, each shell must be located successively further from the solute than the previous one.

- **VShell[1,2,3]** (vshell1, ...): The distances at which to locate additional shells of exact SRFP evaluation points. The SRFP is always evaluated, exactly, at the solute atom sites.
- **TimeStep** (dt): The simulation time step. This is read in the &ipolq namelist just as if it were present in the &cntrl namelist, but a value specified in &ipolq overrides the &cntrl setting. Default is 0.001ps, set in &cntrl.
- **MinQWeight** (minqwt): The stiffness of harmonic restraint by which to restrain fitted shell charges to zero. Default 0.01.
- **ModifyQ** (modq): When IPolQ is applied, it is appropriate to hyper-polarize certain molecules in the SRFP calculation. This variable may be specified as many times as necessary, followed by an ambmask string and a real number indicating the new charges to be assigned to all atoms in the mask. For example, fixed-charge water models should have their dipoles increased by an amount equal to the original model's dipole less 1.85 (the dipole of water in vacuum).
- **QuantumPrep** (prepqm): Preparatory call for QM calculations. This variable may be specified as many times as necessary. Each of these calls will be issued, in the order specified, before executing quantum calculations.
- **QuantumClean** (postqm): Post-processing calls for QM calculations. Similar to prepqm directives, called after QM calculations have been completed.
- **QMPackage** (qmprog): The quantum package to use. Supported packages are "gaussian" and "orca".
- **QMPath** (qmpath): Path to the primary QM executable. This path will be tested, taking into account prepqm calls, to be sure that the executable exists prior to running the SRFP calculation.
- **QMInputFile** (qmcomm): The base name of the QM input file. Vacuum and condensed-phase versions will be written with extensions 'vacu' and 'solv', respectively. Default 'IPolQinp'.
- **Maxmemory** (maxcore): The maximum memory that can be allocated to arrays for quantum calculations with Orca, or the maximum total memory that can be allocated for calculations with Gaussian.
- **QMOutputFile** (qmresult): The base name of the QM output file, which is given similar extensions to the input file. Default 'IPolQout'.
- **PointQFile** (ptqfi): The name of the point charges file referenced by orca for including the SRFP into the condensed-phase calculation.
- **QMSignal** (qmflag): The name of the file used to signal slave processes that the QM calculations launched by the master are complete. Default '.mdgx.finqm'.
- **QMTheory** (qmlev): The level of QM theory to use. Default MP2.
- **QMBasis** (basis): The QM basis set to use. Default cc-pvTZ.
- **WorkDirectory** (scrdir): The scratch directory to use during QM calculations. Useful to reduce NFS load. If the directory exists, it will be used but not destroyed following each QM calculation. If the directory does not exist at the start of the run, it will be created and later destroyed.
- **KeepQMInput** (rqminp): Directive to retain QM input files after the run. Default 0 (OFF).
- **KeepQMCheckPt** (rqmchk): Directive to retain QM checkpoint file(s) after the run. Default 0 (OFF).
- **KeepQCloud** (rcloud): Directive to retain the solvent charge density cloud file after the run. Default 0 (OFF).

- **CheckExist** (checkex): Activates safety checks for the existence of QM executables (including electrostatic potential calculators) called at the start of the run. These checks attempt to take into account user-specified preparatory directives (see prepqm above). Default 1 (ON). Set to zero to disable this safeguard, for instance if the checks cannot find the executables but the preparatory directives, when fully implemented, are known to result in success.

- **UElec[X,Y,Z]Bin** (un[x,y,z]): The number of grid points on which to evaluate the electrostatic potential, in the X direction. Grid dimensions in Y and Z are set by similar variables.

- **UElec[X,Y,Z]Spc** (uh[x,y,z]): The grid spacing of the electrostatic potential grid in the X direction. The grid is always rectilinear. Spacings in Y and Z are set by similar variables.

- **CenterGrid** (cengrid): Directive to center the electrostatic potential grid on the location of the molecule stored in mdgx. The default behavior varies with each quantum package: 'orca' activates centering on the molecule whereas 'gaussian' calls for centering on the origin, as Orca does not reposition the molecule in its output but Gaussian will place the molecule in a 'Standard Orientation' and leave it there in the output and checkpoint files used for electrostatic potential calculations.

- **FormChkPath** (fmpath): Path to the program called for converting binary checkpoint files into formatted checkpoint files. Needed only if the QM program is 'gaussian'.

- **UEvalPath** (uvpath): Path to the program called for evaluating the electrostatic potential grid.

- **GridFile** (grid): Base name of the electrostatic potential grid to be written. As with QM input and output, this base name is appended 'vacu' or 'solv' for vacuum and condensed-phase calculations.

26.6. Restrained Electrostatic Potential Fitting in mdgx

Because of the extensive capabilities for adding virtual sites, *mdgx* also contains an internal means of assigning charges to them. The Restrained Electrostatic Potential (RESP) methodology is the basis for charge assignment based on quantum-mechanical electrostatic potential data, but the details differ somewhat from the implementation in *antechamber*.

The basic concept of fitting charges to reproduce the electrostatic potential of a molecule, by finding the solution with least squared error in the presence of restraints, is carried over from the original Kollmann RESP. However, instead of Langrangian constraints, equivalent charges are unified as single variables in the fit, and penalty functions are added to the fitting matrix to enforce total charge constraints. Where *mdgx* excels is in the control it gives the user over what fitting data will be used. Rather than relying on a quantum-chemistry package to select a particular surface around a molecule, *mdgx* will read the electrostatic potential due to that molecule on a regular grid and select points from that grid based on a solvent-accessible region determined by the actual Lennard-Jones parameters of the model. Because most solvent models make use of hydrogen atoms with modest or non-existent steric properties, *mdgx* also considers points which may not be accessible to the solvent probe but might be accessible to a hydrogen atom connected to that probe. *mdgx* will read a **prmtop** describing the system and also, if required, a Virtual Sites rule file, so that partial charges may be fitted for any virtual sites that the user wishes to add. Once fitting is complete, *mdgx* can return a new Virtual Sites rule file that will apply the fitted charges to the original **prmtop** in future simulations.

Fitting is called by its own separate &fit namelist, and triggers a distinct run mode in the sense that the program will terminate after the fit is complete. The options available in the &fitq namelist include (shorthand aliases in parentheses):

- **RespPhi** (resp): File names and numerical weight of an electrostatic potential to use in fitting. The format is <string1> <string2> <real1>, where string1 is a Gaussian cubegen format file specifying the electrostatic potential and molecular coordinates and Z-numbers appropriate to the topology specified by string2 and real1 is the numerical weight to be given to this conformation in the fit.

- **IPolQPhi** (ipolq): File names and numerical weight of a pair of electrostatic potentials to use in IPolQ fitting. The format is <string1> <string2> <string3> <real1>, where string1 and string2 are Gaussian cubegen format files relating to the system in vacuum and in a condensed-phase environment (see also the section on Implicitly Polarized Charge creation). Note that the molecular coordinates in both cubegen files must match. As in the resp variable format, string3 is the topology, and real1 is the numerical weight of this conformation.

- **EPRules** (eprules): If specified, *mdgx* will output all fitted charges in the form of a Virtual Sites rule file, which can be given as input to subsequent simulations to modify the original **prmtop** and apply the fitted charge model.

- **ConfFile** (conf): If specified, *mdgx* will output the first molecular conformation, complete with any added virtual sites, in PDB format for inspection. This is useful for understanding exactly what model is being fitted.

- **TotalQ** (qtot): The total charge constraint in units of the proton charge; the sum of all fitted charges is required to equal this value. Default 0.0.

- **MinimizeQ** (minq): Restrain the charges of a group of atoms to zero by the weight given in minqwt. The groups are specified in ambmask format.

- **EqualizeQ** (equalq): Restrain the charges of a group of atoms to have the same values. Groups are specified in ambmask format.

- **MinQWeight** (minqwt): Weight used for restraining values of charges to zero; as more and more fitting data is included (either through a higher sampling density of the electrostatic potential due to each molecular conformation or additional molecular conformations) higher values of minqwt may be needed to keep the fitted charges small. However, with more data the need to restrain charges may diminish as well.

- **FitPoints** (nfpt): The number of fitting points to select from each electrostatic potential grid. The points nearest the molecule, which satisfy the limits set by the solvent probe and point-to-point distances as defined below, will be selected for the fit. Default 1000.

- **ProbeSig** (psig): The Lennard-Jones σ parameter of the solvent probe. Default 3.16435 (TIP4P oxygen).

- **ProbeEps** (peps): The Lennard-Jones ε parameter of the solvent probe. Default 0.16275 (TIP4P oxygen).

- **ProbeArm** (parm): The probe arm; points on the electrostatic potential grid that would be inaccessible to the solvent probe may still be included in the fit if they are within the probe arm's reach. Default 0.9572Å (TIP oxygen-hydrogen bond distance).

- **StericLimit** (pnrg): The maximum Lennard-Jones energy of the solvent probe at which a point will qualify for inclusion in the fit. Default 3.0 kcal/mol.

- **Proximity** (flim): The minimum proximity of any two points to be included in the fit. Default 0.4Å.

- **HistogramBin** (hbin): If hist is specified, *mdgx* will print a histogram reporting the number of fitting points falling within any particular distance of some atom of the molecule. This parameter controls the discretization of the histogram.

- **MaxMemory** (maxmem): Because fitting matrices can become very large in some cases (in particular, those involving multiple systems with correlated partial charges), *mdgx* offers this parameter as a safeguard against creating a matrix that may inadvertently take up too much memory. Values for this argument may be integers, or integers followed immediately (no spaces) with terms such as "GB," "Mb," or "kB" (case-insensitive) for giga/mega/kilo bytes. Default 1GB.

- **Verbose** (verbose): Unless set to zero by the user, *mdgx* will print periodic updates and record milestones from the fitting run in terminal output.

An example of a &fit namelist is given below. In this particular problem, ECl2 and ECl3 were the names of virtual sites not in the original topology file but specified by a Virtual Sites rule file.

```
&fit
  RespPhi   Conf12/pcm12.cube,
  RespPhi   Conf13/pcm13.cube,
  RespPhi   Conf14/pcm14.cube,
  pnrg    2.0,
  nfpt    15000,
  minqwt  175.0,
  EqualizeQ '@H1,H2'
  EqualizeQ '@Cl2,Cl3'
  EqualizeQ '@ECl2,ECl3'
  MinimizeQ = '@E*'
  EPRules     frag.xpt
  ConfFile    f6xp.pdb
&end
```

Virtual site constructions have strong support in *mdgx* to rapidly translate between an imagined model and a practical simulation.

26.7. Bonded Term Fitting in mdgx

Having the capabilities to read multiple topologies and coordinate sets, compute energies, and to optimize parameter sets made a bonded parameter fitting module a natural extension of *mdgx*. Like the RESP fitting module, the bond parameter fitting routines can read multiple systems and conformations and determine the best overall values for harmonic bond, harmonic angle, and torsion Fourier series appearing in multiple contexts. The while the RESP module is limited to 512 systems and conformations and makes its fitting matrices based on thousands of data points from each one, there is no practical limit to the number of systems and conformations that the bond parameter fitting module can muster, although it seeks only to make the total internal energy of each conformation match a single target value (presumably obtained from quantum mechanics). This duplicates some functionality in the *paramfit* program described in Chapter 3, but with improved capabilities for large data fitting problems. Results are written to several different files: the **forcedump** file (-d option on the command line or in the &files namelist) stores fitted parameters in the standard Amber parameter file format (i.e. parm99.dat), **mdout** provides extensive analysis of the fit and sampling of each fitted parameter in the data set, and the *accrep* option described in the table below creates a complete report of the correlations, system by system, if requested.

Bonded term fitting is called by including the &param namelist in an inpue file. If detected, this namelist will send *mdgx* into a distinct run mode for parameter optimization. The goal is to take in parameter files, topologies, coordinates, and quantum single point energies, then organize this data. The first thing *mdgx* does is trace all parameters back to their sources in the parameter files (i.e. parm10.dat) and then determine which parameters are open for optimization. Then, *mdgx* compute molecular mechanics energies of all the molecular systems and conformations (as if they were isolated in the gas phase) and subtracts off the portion due to non-optimizable sources. The goal is then to make the *relative* potential energy surfaces of the systems as close as possible to the corresponding quantum energy surfaces. The options available in the &param namelist include (shorthand aliases in parentheses):

- **System** (sys): A fitting data point. This keyword must be followed by three items: the name of a topology file, the name of a corresponding coordinate file, and the energy of this system in the stated conformation.

- **FitBonds** (bonds): Requests a linear least-squares fit for bond stiffnesses in the system.

- **FitAngles** (angles): Requests a linear least-squares fit for angle stiffnesses in the system.

- **FitTorsions** (torsions): Requests a linear least-squares fit for torsion stiffnesses in the system.

- **FitB** (fitb): Request that a specific bond parameter be included in the linear least-squares fitting.
- **FitA** (fita): Request that a specific angle parameter be included in the linear least-squares fitting.
- **FitH** (fith): Request that a specific torsion parameter be included in linear least-squares fitting.
- **FitBondEq** (bondeq): Requests that bond equilibrium constants be fitted alongside their spring constants.
- **FitAnglEq** (angleq): Requests that angle equilibrium constants be fitted alongside their spring constants.
- **FitLJ14** (fitscnb): Requests a linear least-squares fit for Lennard-Jones 1:4 scaling factors.
- **FitEE14** (fitscee): Requests a linear least-squares fit for electrostatic 1:4 scaling factors.
- **ReportAll** (repall): Flag to activate output of all parameters encountered during the fitting procedure, including those that were not adjusted by the fit but nonetheless contributed to the molecular mechanics energies. Default is 1 (write all parameters to the Amber parameter file), appropriate for creating a parm##.dat file to specify a new force field. Set to 0 to create files more akin to frcmod files.
- **ShowProgress** (verbose): Alert the user as to the progress of the fitting procedure. Runs involving thousands of molecular conformations and hundreds of parameters can generally be completed in a few minutes. Default is 1 (ON). Set to zero to suppress output.
- **ElimOutliers** (elimsig): Flag to activate removal of molecular conformations whose energies are far outside the norm for other conformations of the same system. Default 0 (do not remove outliers).
- **ConfTol** (ctol): Tolerance for deviation from the mean energy value, specified as a function of the standard deviation for all conformations of the same system. Conformations of a system which exceed this threshold will be reported if verbose is set to 1, and removed from consideration if elimsig is set to 1. Default 5.0 sigmas.
- **EnergyUnits** (eunits): Units of the target energy values. Default Hartrees. Acceptable values include Hartree/Atomic, kJ/kilojoules, and j/joules. Case insensitive.
- **AccReport** (accrep): Accuracy report on the fit. Contains extensive analysis on the resulting parameters, in MatLab format.
- **ParmTitle** (title): Parameter file title. This is not a file name, but rathr the title appearing on the first line of the printed file named by the `-d` command line / &files namelist argument.
- **Vdw14Fac** (scnb): Sets a universal 1:4 scaling factor for van-der Waals interactions. Use this input to change the scaling on all systems simultaneously.
- **Elec14Fac** (scee): Sets a universal 1:4 scaling factor for electrostatic interactions. Use this input to change the scaling on all systems simultaneously.
- **BondRest** (brst): General value for harmonic restraints on bond stiffness constants.
- **AngleRest** (arst): General value for harmonic restraints on angle stiffness constants.
- **DihedralRest** (hrst): General value for harmonic restraints on angle stiffness constants.
- **RestrainB** (sbrst): Applies a specific restraint stiffness to the value of a fitted bond, equivalent to changing brst for that bond alone. This command takes subdirectives of atom type names, plus 'Keq' and 'Leq' (each followed by a positive real number to denote the target stiffness and equilibrium bond length, respectively). These subdirectives may be given in free format.
- **RestrainA** (sarst): Applies a specific restraint stiffness to the value of a fitted angle, equivalent to changing arst for that angle alone. This command takes subdirectives similar to sbrst, although 'Leq' corresponds to the equilibrium angle rather than length.

- **RestrainH** (shrst): Applies a specific restraint stiffness to the value of a fitted torsion amplitude, equivalent to changing hrst for that torsion alone. This command takes subdirectives (in free format) of atom types, 'period' or 'per' followed by a real number for the periodicity, 'weight' or 'rwt' followed by a real number for the restraint strength (which scales just like hrst), and 'target' or 'trg' followed by a real number if the target value of the restraint is non-zero.
- **BondCoupling** (brstcpl): General value for pricing 1A changes in the fitted equilibrium constant with kcal/mol-A^2 changes in bond spring constants. The default is 5000.0, which penalizes 50 kcal/mol changes in the stiffness at the same rate as 0.01A changes in the equilibrium value. Only relevant with FitBondEq = 1.
- **AngleCoupling** (arstcpl): General value for pricing 1-degree changes in the fitted equilibrium constant with kcal/mol-rad^2 changes in angle spring constants. The default is 114.59, which penalizes 2 kcal/mol changes in the stiffness at the same rate as 1 degree changes in the equilibrium value. Only relevant with FitAnglEq = 1.
- **BondBasisSep** (lpost): Distance form the original bond equilibrium length to place the equilibrium values of either of two basis functions for fitting a new bond term.
- **AngleBasisSep** (thpost): Distance from the original angle equilibrium length to place the equilibrium values of either of two basis functions for fitting a new angle term.
- **Spectrum** (spectrum): Request that a particular bond, angle, or dihedral from among the adjustable parameters be sampled near various values along a specified range, or otherwise included as a reoptimizable variable while others are resampled. This keyword invokes its own sort of namelist: the sub-directives can be given in any order, but they must all be given on the same line. Words that are not explicitly sub-directives or values following them may be counted as atom types, so long as they have fewer than four characters, until four such types are catalogged. Sub-directives include retain (parameters matching this request are reoptimized but not resampled), sample (parameters matching this request are resampled), order (followed by a value, 2 = bonds, 3 = angles, 4 = torsions), min and max (followed by values, the resampling range limits), spc (the resampling discretization), and break (stop adding new sub-directives to this spectrum command). This is an experimental feature, but may be useful for generating multiple force field candidates with subtly different behavior around the optimal data fit.

The data fitting capabilities in *mdgx* focus on a single linear least-squares problem, unless the experimental "spectrum" option is invoked, in which case a series of linear least squares problems are solved. This is a very active area of development in *mdgx*, and will continue to gain new features and capailities in future patches, with the goal of leveraging high-performance computing to deliver robust parameter development to novice users.

26.8. Configuration Sampling

While *mdgx* provides lots of options for its force field applications, the charge and bonded parameter development are little more than big data fitting problems. They rely on a finely sampled and complex data set, and generating such a thing is at tedious process, prone to fatigue and human error. The &configs namelist in mdgx is designed to streamline this process by energy-minimizing hundreds to thousands of instances of a single structure, subject to different constraints, *en masse*. In principle, everything that this module does could be accomplished with a shell script executing *sander* or *pmemd*, but the execution would take tens to hundreds of times longer and things that are easily done with *mdgx* &configs would take complex shell scripts. Even then *mdgx* performs analyses on the data set as a whole that help users to understand whether the data set is suitable for quantum calculations, and if not, how to improve it. With a sander script, creating 500 energy-minimized conformations of a drug molecule that sample rotation around three critical dihedrals might take an hour, and the process may have to be repeated several times to ensure that the restraints are sufficiently stiff or that other desgrees of freedom are properly relaxed or randomized. With *mdgx* &configs, the same process will take about a minute, making it possible to set up higher quality data sets in a single sitting.

To support the operations of configuration sampling, several *mdgx* inputs have been ungraded to support directories and regular expressions rather than simply the names of individual files. A single structure can serve as input to the configurations module to make hundreds of copies, but a trajectory, list of trajectories, or directory containing any number of single-frame files can also serve. In this manner, one structure can become a thousand, or the same minimization protocol can be applied to a thousand conformations of the same structure.

Available options for the &configs module fall into several categories. The minimization protocol itself is guided by parameters similar to *sander* inputs. Additionally, *mdgx* provides a feature for “shuffling” results and attempting additional minimizations. This helps solve the problem of escaping local minima without resorting to much more costly simulations with temperature to jostle small molecules around.

- **Verbose** (verbose): Sets the verbosity level (0 is silent, 1 will give frequent updates on the command line.
- **Replicas** (count): The number of configurations to generate, if starting from a single configuration in inpcrd or restart format.
- **MaxCycles** (maxcyc): The maximum number of line minimization steps to attempt in any one round of energy minimization.
- **SDSteps** (ncyc) The number of steepest descent line minimization steps to perform before switching to a conjugate gradient method. As with the eponymous keywords in sander, ncyc must be less than or equal to maxcyc.
- **ExclTableSize** (exclmax): The maximum number of atoms for which a table of non-bonded scaling factors will be kept. For small systems, it is faster to pre-calculate whether non-bonded interactions will be excluded or attenuated and store these values in a matrix. However, this is memory-intensive and will trash the cache for larger systems. In those cases it is better to store a different sort of data structure that will quickly determine whether two atoms constitute an exclusion.
- **ForceConverge** (frctol): Convergence criterion for the optimization. This is a quantity of force--if forces on all particles have lower magnitude than this value, the energy optimization for that configuration will be deemed converged.
- **StepConverge** (steptol): Convergence criterion for the optimization. This is a quantity of distance--if the movement of all particles along the current force vector is driven lower than this value, the energy optimization for that configuration will be deemed converged.
- **InitialStep** (step0): Initial step size for the energy optimization. This is a quantity of distance: the total magnitude of the initial step along the first computed force vector, that is the square root of the sums of squares of the displacements of all particles from their original positions, will be equal to this number (default 0.01A). The step size will be iteratively changed throughout optimization and will be tailored to each configuration.
- **ShuffleCount** (nshuffle): The number of times to restart energy minimizations towards the specified restraint targets using different initial states.
- **ShuffleStyle** (shuffle): The type of shuffling to perform if nshuffle > 0. Available methods include "bootstrap" (new initial states will be assigned randomly from existing solutions, with replacement--one solution can serve as the initial state for more than one configuration), "jackknife" (the default--each existing solution will be assigned as the initial state for energy reoptimization to one and only one configuration), and "proximity" (every solved configuration will be evaluated in terms of the restraint targets of every other, and new initial states will be randomly chosen from among solutions whose restraint energies are within a certain threshold of the MINIMUM energy found for any existing solution with respect to the particular restraint targets of a given configuration.
- **ProximateNrg** (eprox): Threshold energy for taking existing solutions as the intial states for new attempts at energy minimization if using "proximity" reshuffling (default 5.0 kcal/mol).

- **ReplacementTol** (erep): Threshold for accepting a new solution based on a different initial state. The new solution must supplant the energy of the existing one by at least this amount. Default 1.0e-4 kcal/mol.
- **Direction** (shfdir): The direction to replace energies when reshuffling energy optimizations. Choices are "up" and "down". Default is "down," but replacement can be made to move the energies upwards, finding new local minima with higher overall energies.

The &configs module performs energy minimization subject to NMR-like restraints and supports multiple sampling strategies for each of them. Each restraint and sampling strategy commands contains its own vocabulary and multiple descriptors.

- **RandomSample** (random): Perform random sampling within a range. This keyword must be followed by a series of commands, all on the same line of the input file, but the order of the sub-commands is flexible. After seeing this keyword, mdgx will search the remainder of the line until it hits another keyword from the &configs namelist; until then it will associate any input it finds with the previous "random" or "RandomSample" keyword. The range of sampling in this case is absolute: a flat-bottom harmonic potential will be constructed, centered on the spot randomly chosen between the limits "min" and "max", or given between two { } braces. To specify that all configurations be restrained towards a single target value, the keyword "center" may be used in place of "min", "max", or { }. The potential shall be flat up to a distance "fbhw" (flat bottom half width) from the center, and thereafter rise quadratically with a coefficient "Krst" (stiffness constant K of the restraint) over a length specified by the "quadratic" keyword, or up to a point at which the restraint force would reach a limit given by the "Ftop" keyword. Beyond this limit, the force will be clamped and the restraint potential will be effectively linear, which helps to ensure that restraints to positions far from the initial configuration do not break things like chirality. Because it is more intuitive to specify a maximum restraint force than a quadratic window, "Ftop" will take priority over "quadratic" if both keywords are given. The defaults are to have 64 kcal/mol restraints applied after a 0.5A flat bottom half width, topping out at 32 kcal/mol-A applied force.
- **GridSample** (uniform): Perform sampling on regular intervals within a range. All of the keywords from RandomSample apply here as well.
- **RandomPerturb** (rpert): Perform sampling on regular intervals within a range based on the arrangement of atoms in each initial structure. All of the keywords from RandomSample apply here as well, except that the range now specifies minimum and maximum values relative to the initial arrangement of atoms. If multiple initial structures are read in, this will perturb each of them by similar random amounts.
- **GridPerturb** (gpert): Perform sampling on regular intervals within a range based on the initial arrangement of atoms. This is to RandomPerturb as GridSample is to RandomSample.
- **LinkOperationsCombine** (combine): Combine two operations involving grid-based, interval sampling. Without any such combinations, the interval sampling restraints in each configuration will march from one end of their respective ranges to the other, in unison--this will generate a line of configurations in the multi-dimensional space defined by each restrained coordinate. To sample two or three dimensions of the space simultaneously at regular intervals, combine the operations. Up to three operations may be combined. For N combined operations, mdgx will take the Nth root of the total number of configurations and take this many samples along each of the combined restraint dimensions.
- **MovingAtoms** (belly): Make only the atoms in the given ambmask string movable during geometry optimization.

The &configs module supports multiple output formats for the minimized structures, including the well known PDB format and outputs that can serve as input files for some popular quantum packages.

- **ShowOrigins** (showorig): Flag to have mdgx show the original files for each configuration that it solves. If all configurations start from a single state in a single file, the default behavior is to withhold this reporting. However, if there are many files, the origin of each configuration may not be so obvious, and while mdgx does attempt to alphabetize and organize long lists of files arising from directory searches or regular expressions the evolution of molecular configurations may be of interest. In these cases the default behavior is to report the origins of each configuration.

- **OutputBase** (outbase): The bases of the output file names for configurations. The format will be <base><number>.<suffix>. Multiple strings may follow this keyword, so long as they are all on the same line. Each string provided will be matched with a suffix and a style provided, in the order each is given.

- **OutputSuffix** (outsuff): Suffixes of the output file names for printed configurations.

- **OutputType** (write): The type of output to write, options being "CRD" (old Amber .crd format trajectory), "CDF" (Amber netCDF trajectory), "INPCRD" (Amber ascii 7-decimal place inpcrd file for individual configurations), "PDB" (PDB format, with descriptions of the way the configuration was generated in the REMARK section), and "ORCA", "GAUSSIAN", "MOLPRO", and "GAMESS" for input files to various quantum packages. If trajectories are being written, all configurations that pass sanity checks will be printed to the file. For the other formats, individual configurations will be printed to separate files. More than one type of output may be written after creating a set of configurations.

Sanity checking is an essential part of data set creation. Because all mdgx outptus are already energy minimized with respect to an input force field, they are already mostly “sane.” However, with any minimization there can be traps, local minima, and residual strain that should be considered before submitting the configuration to quantum methods or using it as input data for making a force field. *mdgx* will automatically check for some common problems, decline to print structures that have them, and report what went wrong so that the configuration sampling can be repeated for better results.

- **BondStrain** (bstrain): The maximum bond strain (according to the input force field, as given in the topology file) that will be tolerated in any configuration that is to be printed.

- **AngleStrain** (astrain): The maximum bond angle strain (according to the input force field, as given in the topology file) that will be tolerated in any configuration that is to be printed.

- **StrainLimit** (strainlim): The maximum restraint penalty that will be tolerated in any configuration. Note that, for any of these sanity checks, convergence of the energy minimization is NOT an automatic fail--it will simply be noted in the report file summarizing the process. Rather, the sanity checks pertain to features of the structures that appear well outside the applicable range of molecular mechanics functions.

When the output structures are given as inputs for quantum calculations, there are other considerations for running the QM program itself. While advanced use of the quantum programs may require post-processing the files with a shell script, *mdgx* does interpret some basic run parameters and incorporate them into its results.

- **MaxMemory** (maxcore): When ordering mdgx to print configurations as input files to quantum packages, this states how much memory should be available for QM calculations.

- **CPUCount** (ncpu): The number of CPUs to apply in each QM calculation.

- **Multiplicity** (spin): The multiplicity to assign to this system (in all its configurations) for quantum calculations. mdgx is not able to calculate this on its own.

- **QMTheory** (qmlev): The level of theory to apply in quantum calculations.

- **QMBasis** (basis): The basis set to apply in quantum calculations.

- **Checkpoint** (chk): The checkpoint file to write if using Gaussian for QM calculations.

26.9. Thermodynamic Integration

A rudimentary implementation of thermodynamic integration is available in *mdgx*. This facility is not fully developed, but does permit users to test changes in hydration free energy or other consequences of new charge models, such as those that include virtual sites. The only significant similarity to *sander* is that there are two trajectories propagated simultaneously using a mixture of the forces obtained at each endpoint; otherwise the implementation is very different. In *mdgx*, both trajectories are propagated by the same processor, so it is feasible to run TI in serial mode, without a parallel build. A single input file carries all the necessary information for a thermodynamic integration run, including the names of the topologies describing the initial and final states of the system and the path for changing between them. A single output file contains all the relevant information concerning the energies of the system at each endpoint and the derivative of the potential with respect to the coupling parameter λ.

Parameters specific for thermodynamic integration in mdgx include (shorthand aliases in parentheses):

- **RunTI** (icfe): Flag to turn on thermodynamic integration (default 0, set to 1 to activate)
- **MixOrder** (klambda): The exponent on terms involving the coupling parameter λ (see Section 21.1; default 1)
- **MixFactor** (clambda): The value of the coupling parameter λ (default 0.0)
- **SynchTI** (nsynch): Frequency at which to explicitly synchronize the two trajectories. In principle, this should never be necessary but to prevent some corner case from occurring the nsynch is set to 1000 steps by default. A brief report of activity required to synchronize coordinates appears in the output file every time this routine is called.

The mdgx program can accept up to two topologies, for the initial and final states of the system. The topologies are specified by the argument "-p#" on the command line or the arguments "Topology#" or "-p#" in the **mdin** input file, where # is blank, 1, or 2. A blank value of # corresponds to 1. Different Virtual Site rules files may be specified for each topology with the "-xpt#" option on the command line and the "-xpt#" or "EPRules#" options in the **mdin** file. Assigning the same topology file to both -p1 and -p2 parameters but assigning a Virtual Sites rule file to one of them is a way to test the energetic consequences of changing the charge model found in some standard force field to one that includes new virtual sites.

Thermodynamic integration routines in mdgx can handle topologies of different numbers of atoms. Unique atoms are each endpoint are considered points with mass but no other properties at the other endpoint. This functionality is not yet mature, so "experts only!" With the future addition of soft-core repulsive potentials for smooth growth and removal of atoms, this functionality will become more robust and accessible to end users.

26.10. Future Directions and Goals of the mdgx Project

While it does draw on adaptations of some code found in the *sander* program, *mdgx* is not a re-implementation of a subset of *sander*'s functionality. Many of the algorithms used by *mdgx* differ from those used by *sander* and *pmemd*, including the velocity version of the Verlet integrator, the domain decomposition for nonbonded interactions, and an atomic, as opposed to a molecular, virial calculation (if the virial is computed at all). This independence of the *mdgx* program may create some difficulties when trying to compare *mdgx* results from simulations to *sander* and *pmemd*, but efforts are being made to unify the input and output conventions of each program. The simple C implementation in *mdgx* should be adaptable and expandable, and the nexus of capabilities for reading topologies, computing forces and energies, executing dynamics, and fitting parameters make the program a useful tool for testing and comparing new algorithms.

Since the version 14 release of AmberTools, *mdgx* offers parallel scaling to 8 or more CPUs, for a roughly five- to six-fold speed advantage over its serial implementation. This level of parallelism is sufficient for accelerating the molecular dynamics portion of IPolQ charge derivations, and will likely remain in place for the foreseeable future.

Part V.

Analysis of simulations

27. mdout_analyzer.py and ambpdb

mdout_analyzer.py is a simple script designed to help you rapidly parse and analyze the energy components printed in the output files from *sander* and *pmemd*. It requires that the *numpy* and *matplotlib* packages be installed. The *scipy* Python package is also required when plotting smoothed histograms using kernel density estimates. You can use it as follows:

```
mdout_analyzer.py <mdout1> <mdout2> <mdout3> ... <mdoutN>
```

Where each mdout file is combined into a single data set. A GUI window will open up with buttons for every energy component parsed from the mdout file followed by a button for each type of graphical analysis you can do on the data shown below.

A second window has options to control how the graphs will appear. Help is available in the <Help> menu at the top of the main window. Note, mdout files must be from the same type of simulation (or at least have all of the same energy components printed inside) in order to be combined.

Right-clicking on each energy button brings up a little window describing what that energy term is.

27.1. ambpdb

NAME ambpdb - convert amber-format coordinate files to pdb format

SYNOPSIS

```
ambpdb [ -p prmtop-file ] < AmberRestartFile
ambpdb [ -p prmtop-file ] -c coordinate-file
```

Additional Options:

```
[ -tit title ] [ -pqr|-mol2] [ -aatm ] [-bres ] [-noter] [-offset #] [ -ext ]
```

ambpdb is a filter to take a coordinate "restart" file from an AMBER dynamics or minimization run and prepares a pdb-format file (on STDOUT). The program assumes that a *prmtop* file is available, from which it gets atom and residue names. Note: starting with AmberTools15, ambpdb can convert *any* coordinate file format that CPPTRAJ can read using the `'-c'` flag. Either an Amber restart file must be directed in via STDIN or a file with '-c' must be specified.

OPTIONS

-h	Print a usage summary to the screen.
-p	Specify the Amber topology file to use (if not specified will look for file named “prmtop”).
-c	Instead of reading an Amber restart from STDIN, specifies file to read coordinates from; can be any format that CPPTRAJ can read.
-tit	The title, if given, will be output as a REMARK at the top of the file. It should be protected by quotes or double quotes if it contains spaces or special characters.
-pqr	If *-pqr* is set, output will be in the format needed for the electrostatics programs that need charge and radius information.

-mol2	creates a TRIPOS mol2 file with all of the residues and bond information present in the topology file.
-aatm	This switch controls whether the output atom names follow Amber or Brookhaven (PDB) formats. With the default (when this switch is not set), atom names will be placed into four columns following the rules used by the Protein Data Base in Version 3.
-bres	If *-bres* (Brookhaven-residue-names) is not set (the default), Amber-specific atom names (like CYX, HIE, RG5, etc.) will be kept in the pdb file; otherwise, these will be converted to PDB-standard names (CYS, HIS, G, in the above example). Note that setting *-bres* creates a naming ambiguity between protonated and unprotonated forms of amino acids. If you plan to re-read the pdb file back into Amber programs, you should use the default behavior; for programs that demand stricter conformance to Brookhaven standards, set *-bres*.
-noter	If *-noter* is set, the output PDB file not include TER cards between molecules. Otherwise, TER cards will be added whenever there is not bond between adjacent residues. Note that this means there will be a TER card between each water molecule, for example, unless *-noter is set*. The PDB is idiosyncratic about TER cards: they are generally present between separate protein chains, but generally not present between cofactors or solvent molecules. This behavior is not mimicked by *ambpdb*.
-offset	If a number is given here, it will be added to all residue numbers in the output pdb file. This is useful if you want the first residue (which is always "1" in an Amber prmtop file, to be a larger number, (say to more closely match a file from Brookhaven, where initial residues may be missing). Note that the number you provide is one less than what you want the first residue to have. Residue numbers greater than 9999 will not "fit" into the Brookhaven format; ambpdb actually prints mod(resno,10000); that is, after 9999, the residue number re-cycles to 0.
-ext	This tells ambpdb to use any extended PDB info present in prmtop-file (from using e.g. the ’addPDB’ command from *parmed*).

28. cpptraj

Cpptraj[556] (the successor to *ptraj*) is the main program in Amber for processing coordinate trajectories and data files. *Cpptraj* has been developed to be almost completely backwards-compatible with *ptraj* input. In general, if a command in *ptraj* has been implemented in *cpptraj* it should produce similar results, although the output format may be different. A notable exception is the ***hbond*** command, for which the syntax is quite different (see 28.9.34 on page 613 for details). *Cpptraj* is at least as fast as *ptraj* was, and is in many cases significantly faster, particularly when processing NetCDF trajectories. In addition, several actions have been parallelized with OpenMP to take advantage of multi-core machines for even more speedup (see section 28.1.6.2 on page 555), and ensembles of trajectories can be processed in parallel with MPI (see section 28.1.6.1 on page 554). Currently the only functionality from *ptraj* not implemented in *cpptraj* are certain clustering algorithms.

Here are several notable features of *cpptraj*:

1. Trajectories with different topologies can be processed in the same run.
2. Several actions/analyses in *cpptraj* are OpenMP parallelized; see section 28.1.6.2 for more details.
3. Trajectory and ensemble reads can be MPI parallelized.
4. Almost any file read or written by *cpptraj* can be compressed (with the exception of the NetCDF trajectory format). So for example gzipped/bzipped topology files can be read, and data files can be written out as gzip/bzip2 files. Compression is detected automatically when reading, and is determined by the filename extension (.gz and .bz2 respectively) on writing.
5. The format of output data files can be specified by extension. For example, data files can be written in xmgrace format if the filename given has a '.agr' extension. A trajectory can be written in DCD format if the '.dcd' extension is used.
6. Multiple output trajectories can be specified, and can be written during action processing (as opposed to only after) via the ***outtraj*** command. In addition, output files can be directed to write only specific frames from the input trajectories.
7. Multiple reference structures can be specified. Specific frames from trajectories may be used as a reference structure.
8. The ***rmsd*** action allows specification of a separate mask for the reference structure. In addition, per-residue RMSD can be calculated easily.
9. Actions that modify coordinates and topology such as the ***strip/closest*** actions can often write an accompanying fully-functional stripped topology file.
10. Users usually are able to fine-tune the output format of data files declared in actions using the "**out**" keyword (for example, the precision of the numbers can be changed). In addition, users can control which data sets are written to which files (e.g. if two actions specify the same data file with the 'out' keyword, data from both actions will be written to that data file).
11. Users can manipulate data sets using mathematical expressions (with some limitations), see 28.3.2 on page 559 for details.

28.0.1. Manual Syntax Format

The syntax presented in this manual uses the following conventions:

<> Denotes a variable.

[] Denotes something is optional.

{|} Denotes several choices separated by the '|' character; one of the choices must be specified.

... Denotes the preceding option can be repeated.

Everything else is as printed.

28.1. Running Cpptraj

Cpptraj can be run in either "interactive mode" or in "batch mode".

28.1.1. Command Line Syntax

```
cpptraj [-p <Top0>] [-i <Input0>] [-y <trajin>] [-x <trajout>]
        [-c <reference>]
        [-h | --help] [-V | --version] [--defines] [-debug <#>]
        [--interactive] [--log <logfile>] [-tl]
        [-ms <mask>] [-mr <mask>] [--mask <mask>] [--resmask <mask>]
```

-p <Top0> `Load <Top0> as a topology file. May be specified more than once.`

-i <Input0> `Read input from <Input0>. May be specified more than once.`

-y <trajin> `Read from trajectory file <trajin>; same as input 'trajin <trajin>'.`

-x <trajout> `Write trajectory file <trajout>; same as input 'trajout <trajout>'.`

-c <reference> `Read <reference> as reference coordinates; same as input 'reference <reference>'.`

-h | –help `Print command line help and exit.`

-V | –version `Print version and exit.`

–defines `Print compiler defines and exit.`

-debug <#> `Set global debug level to <#>; same as input 'debug <#>'.`

–interactive `Force interactive mode.`

–log <logfile> `Record commands to <logfile> (interactive mode only). Default is 'cpptraj.log'.`

-tl `Print length of trajectories specified with '-y' to STDOUT. The total number of frames is written out as 'Frames:  <X>'`

-ms <mask> `Print selected atom numbers to STDOUT. Selected atoms are written out as 'Selected= 1 2 3 ...'`

-mr `<mask> :  Print selected residue numbers to STDOUT. Selected residues are written out as 'Selected= 1 2 3 ...'`

–mask <mask> `Print detailed atom selection to STDOUT.`

–resmask `<mask> :  Print detailed residue selection to STDOUT.`

For backwards compatibility with *ptraj*, the following syntax is also accepted:

```
cpptraj <Top> <Input>
```

Note that unlike *ptraj*, in *cpptraj* it is not required that a topology file be specified on the command line as long as one is specified in the input file with the 'parm' keyword. Multiple topology/input files can be specified by use of multiple '-p' and '-i' flags. All topology and coordinate flags will be processed before any input flags.

28.1.2. Commands

Input to *cpptraj* is in the form of commands, which can be categorized in to 2 types: immediate and queued. Immediate commands are executed as soon as they are encountered. Queued commands are initialized when they are encountered, but are not executed until a Run is executed via a ***run*** or ***go*** command. Actions, Analyses, and Trajectory commands (except ***reference***) are queued commands; however, they can also be run immediately via commands such as ***crdaction***, ***runanalysis***, ***loadcrd***, etc. See 28.5 on page 564 for more details.

Commands fall into seven categories:

General (Immediate) These commands are executed immediately when entered.

System (Immediate) These are unix system commands (e.g. 'ls', 'pwd', etc).

Coords (Immediate) These commands are used to manipulate COORDS data sets; see 28.5 on page 564 for more details.

Trajectory (Queued) These commands prepare cpptraj for reading or writing trajectories during a Run.

Topology (Immediate) These commands are used to read, write, and modify topology information.

Action (Queued) These commands specify actions that will be performed on coordinate frames read in from trajectories during a Run.

Analysis (Queued) These commands specify analyses that will be performed on data that has been either generated from a Run or read in from an external source.

In addition to normal commands, *cpptraj* now has the ability to perform certain basic math operations, even on data sets. See 28.3.2 on page 559 for more details.

Commands in *cpptraj* can be read in from an input file or from the interactive command prompt. A '#' anywhere on a line denotes a comment; anything after '#' will be ignored no matter where it occurs. A '\' allows the continuation of one line to another. For example, the input:

```
# Sample input
trajin mdcrd # This is a trajectory
rms first out rmsd.dat \
    :1-10
```

Translates to:

```
trajin mdcrd
rms first out rmsd.dat :1-10
```

If in interactive mode, 'help <command>' can be used to get the associated keywords as well as an abbreviated description of the command. Most commands have a corresponding test which also serves as an example of how to use the command. See $AMBERHOME/AmberTools/test/cpptraj/README for more details.

28.1.3. Batch mode

In “batch” mode, cpptraj is executed from the command line with one or more input files containing commands to be processed or STDIN. The syntax of <input file> is similar to that of *ptraj*. Keywords specifying different commands are given one per line. Lines beginning with '#' are ignored as comments. Lines can also be continued through use of the '\' character. This is the only allowed mode for cpptraj.MPI.

28.1.4. Interactive mode

In “interactive mode” users can enter commands in a UNIX-like shell. Interactive mode is useful for running short and simple analyses or for trying out new kinds of analyses. If *cpptraj* is run with ’–interactive’, no arguments, or no specified input file:

```
cpptraj
cpptraj --interactive
cpptraj <parm file>
cpptraj -p <parm file>
```

this brings up the interactive interface. This interface supports command history (via the up and down arrows) and tab completion for commands and file names. If no log file name has been given (with ’–log <logfile>’), all commands used in interactive mode will be logged to a file named ’cpptraj.log’, which can subsequently be used as input if desired. Command histories will be read from any existing logs.

28.1.5. Trajectory Processing “Run”

Like *ptraj*, a trajectory processing “Run” is one of the main ways to run *cpptraj*. First the Run is set up via commands read in from an input file or the interactive prompt. Trajectories are then read in one frame at a time (or in the case of ensemble processing all frames from a given step are read). Actions are performed on the coordinates stored in the frame, after which any output coordinates are written. At the end of the run, any data sets generated are written, and any queued Analyses are performed.

28.1.5.1. Actions and multiple topologies

Since *cpptraj* supports multiple topology files, during a Run actions are set up every time the topology changes in order to recalculate things like what atoms are in a mask etc. Actions that are not valid for the current topology are skipped for that topology. So for example given two topology files with 100 residues, if the first topology file processed includes a ligand named MOL and the second one does not, the action:

```
distance :80 :MOL out D_80-to-MOL.dat
```

will be valid for the first topology but not for the second, so it will be skipped as long as the second topology is active.

28.1.6. Parallelization

Cpptraj has many levels of parallelization that can be enabled via the ’-mpi’, ’-openmp’, and/or ’-cuda’ configure flags for MPI, OpenMP, and CUDA parallelization respectively. At the highest level, trajectory and ensemble reads are parallelized with MPI. In addition, certain time consuming actions have been parallelized with OpenMP and/or CUDA.

Note that any combination of the ’-openmp’, ’-cuda’, and ’-mpi’ flags may be used to generate a hybrid MPI/OpenMP/CUDA binary; however this may require additional runtime setup (e.g. setting OMP_NUM_THREADS for OpenMP) to work properly and not oversubscribe cores.

28.1.6.1. MPI Trajectory Parallelization

Cpptraj has two levels of MPI parallelization for reading input trajectories. The first is for ’***trajin***’ trajectory input, where the trajectory read is divided as evenly as possible among all input frames. For example, if given two trajectories of 1000 frames each and 4 MPI threads, thread 0 reads frames 1-500 of trajectory 1, thread 1 reads frames 501-1000 of trajectory 1, thread 2 reads frames 1-500 of trajectory 2, and thread 3 reads frames 501-1000 of trajectory 2.

The second is for ***’ensemble’*** trajectory input, where the reading/processing/writing of each member of the ensemble is divided up among MPI threads. The number of MPI threads must be a multiple of the ensemble size.

If the number of threads is greater than the ensemble size then the processing of each ensemble member will be divided among MPI threads. For example, given an ensemble of 4 trajectories and 8 threads, threads 0 and 1 are assigned to the first ensemble trajectory, threads 2 and 3 are assigned to the second ensemble trajectory, and so on. When using ensemble mode in parallel it is recommended that the ***ensemblesize*** command be used prior to any ensemble command as this will make set up far more efficient.

In order to use the MPI version, Amber/*cpptraj* should be configured with the '-mpi' flag. You can tell if *cpptraj* has been compiled with MPI as it will print 'MPI' in the title, and/or by calling 'cpptraj —defines' and looking for '-DMPI'.

28.1.6.2. OpenMP Parallelization

Some of the more time-consuming actions/analyses in *cpptraj* have been parallelized with OpenMP to take advantage of machines with multiple cores. In order to use OpenMP parallelization Amber/*cpptraj* should be configured with the '-openmp' flag. You can easily tell if *cpptraj* has been compiled with OpenMP as it will print 'OpenMP' in the title, and/or by calling 'cpptraj —defines' and looking for '-D_OPENMP'. The following actions/analyses have been OpenMP parallelized:

```
2drms/rms2d
atomiccorr
checkstructure
closest
cluster (pair-wise distance calculation and sieved frame restore only)
dssp/secstruct
gist (non-bonded calculation)
kde
mask (distance-based masks only)
matrix (coordinate covariance matrices only)
minimage
radial
replicatecell
rmsavgcorr
spam
surf
velocityautocorr
volmap
watershell
wavelet
```

By default OpenMP *cpptraj* will use all available cores. The number of OpenMP threads can be controlled by setting the OMP_NUM_THREADS environment variable.

28.1.6.3. CUDA Parallelization

Some time-consuming actions in cpptraj have been parallelized with CUDA to take advantage of machines with NVIDIA GPUs. In order to use CUDA parallelization Amber/cpptraj should be configured with the '-cuda' flag. You can easily tell if cpptraj has been compiled with CUDA as it will print 'CUDA' and details on the current graphics device in the title, and/or by calling 'cpptraj —defines' and looking for '-DCUDA'. The following actions have been CUDA parallelized:

```
closest
watershell
```

28.2. General Concepts

28.2.1. Units

Cpptraj uses the AKMA system of units. The execption is time, which is typically expressed in ps (except where noted).

Variable	Unit
Length	Angstrom
Energy	kcal/mol
Mass	AMU
Charge	electron
Time	ps (typically)
Force	kcal/mol*Angstrom

28.2.2. Atom Mask Selection Syntax

The mask syntax is similar to *ptraj*. Note that the characters '`:`', '`@`', and '`*`' are reserved for masks and should not be used in output file or data set names. All masks are case-sensitive. Either names or numbers can be used. Masks can contain ranges (denoted with '`-`') and comma separated lists. The logical operands '`&`' (and), '`|`' (or), and '`!`' (not) are also supported.

The syntax for elementary selections is the following:

:{residue numlist} e.g. ':1-10', ':1,3,5', ':1-3,5,7-9'

:{residue namelist} e.g. ':LYS', ':ARG,ALA,GLY'

@{atom numlist} e.g. '@12,17', '@54-85', '@12,54-85,90'

@{atom namelist} e.g. '@CA', '@CA,C,O,N,H'

@%{atom type name} e.g. '@%CT'

@/{atom_element_name} e.g. '@/N'

<mask><distance op><distance> Selection by distance, see below.

Several wildcard characters are supported:

'*' Zero or more characters.

'=' Same as '*'

'?' One character.

The wildcards can also be used with numbers or other mask characters, e.g. ':?0' means ":10,20,30,40,50,60,70,80,90", ':*' means all residues and '@*' means all atoms.

Compound expressions of the following type are allowed:

```
:{residue numlist | namelist}@{atom namelist | numlist}
```

and are processed as:

```
:{residue numlist | namelist} & @{atom namelist | numlist}
```

e.g. ':1-10@CA' is equivalent to ":1-10 & @CA".

More examples:

:ALA,TRP All alanine and tryptophan residues.

:5,10@CA CA carbon in residues 5 and 10.

:*&!@H= All non-hydrogen atoms (equivalent to "!@H=").

@CA,C,O,N,H All backbone atoms.

!@CA,C,O,N,H All non-backbone atoms (=sidechains for proteins only).

:1-500@O&!(:WAT|:LYS,ARG) All backbone oxygens in residues 1-500 but not in water, lysine or arginine residues.

Distance-based Masks

There are two very important things to keep in mind when using distance based masks:

1. Distance-based masks that update each frame are currently only supported by the ***mask*** action.
2. Selection by distance for everything but the ***mask*** action requires defining a reference frame with ***reference***; distances are then calculated using the specified reference frame only. This reference frame can be changed using the ***activeref*** command.

The syntax for selection by distance is a **<mask>** expression followed by a **<distance operator>** followed by a **<distance>** (which is in Angstroms). The **<distance operator>** consists of 2 characters: '**<**' (within) or '**>**' (without) followed by '**:**' (residues) or '@' (atoms), so '**<:**' means "residues within" etc. For residue based distance selection, if any atom in that residue matches the given distance criterion, the entire residue is selected.

In plain language, the entire distance mask can be read as "Select **<distance operator> <distance>** of **<mask>**". So for example, the mask expression:

```
:11-17<@2.4
```

Means "Select atoms within 2.4 Å distance of atoms selected by ':11-17' (residues numbered 11 through 17). As another example, to strip everything outside 3.0 Å (i.e. without 3.0 Å) from residue 4 using specified reference coordinates:

```
reference mol.rst7
trajin mol.rst7
strip !(:4<:3.0)
```

28.2.3. Ranges

For several commands some arguments are ranges (e.g. 'trajout onlyframes <range>', 'nastruct resrange <range>', 'rmsd perres range <range>'); **THESE ARE NOT ATOM MASKS**. They are simple number ranges using '-' to specify a range and ',' to separate different ranges. For example 1-2,4-6,9 specifies 1 to 2, 4 to 6, and 9, i.e. '1 2 4 5 6 9'.

28.2.4. Parameter/Reference Tagging

Parameter and reference files may be 'tagged' (i.e. given a nickname); these tags can then be used in place of the file name itself. A tag in *cpptraj* is recognized by being bounded by brackets ('[' and ']'). This can be particularly useful when reading in many parameter or reference files. For example, when reading in multiple reference structures:

```
trajin Test1.crd
reference 1LE1.NoWater.Xray.rst7 [xray]
reference Test1.crd lastframe [last]
reference Test2.crd 225 [open]
rms Xray ref [xray] :2-12@CA out rmsd.dat
rms Last ref [last] :2-12@CA out rmsd.dat
rms Open ref [open] :2-12@CA out rmsd.dat
```

This defines three reference structures and gives them tags [xray], [last], and [open]. These reference structures can then be referred to by their tags instead of their filenames by any action that uses reference structures (in this case the RMSD action).

Similarly, this can be useful when reading in multiple parameter files:

```
parm tz2.ff99sb.tip3p.truncoct.parm7 [tz2-water]
parm tz2.ff99sb.mbondi2.parm7 [tz2-nowater]
trajin tz2.run1.explicit.nc parm [tz2-water]
reference tz2.dry.rst7 parm [tz2-nowater] [tz2]
rms ref [tz2] !(:WAT) out rmsd.dat
```

This defines two parm files and gives them tags [tz2-water] and [tz2-nowater], then reads in a trajectory associated with one, and a reference structure associated with the other. Note that in the 'reference' command there are two tags; the first goes along with the 'parm' keyword and specifies what parameter file the reference should use, the second is the tag given to the reference itself (as in the previous example) and is referred to in the subsequent RMSD action.

28.3. Data Sets and Data Files

In *cpptraj*, Actions and Analyses can generate one or more data sets which are available for further processing. For example, the ***distance*** command creates a data set containing distances vs time. The data set can be named by the user simply by specifying a non-keyword string as an additional argument. If no name is given, a default one will be generated based on the action name and data set number. For example:

```
distance d1-2 :1 :2 out d1-2.dat
```

will create a data set named “d1-2”. If a name is not specified, e.g.:

```
distance :1 :2 out d1-2.dat
```

the data set will be named “Dis_00000”.

Data files are created automatically by most commands, usually via the “**out**” keyword. Data files can also be explicitly created with the ***write/writedata*** and ***create*** commands. Data can also be read in from files via the ***readdata*** command. Cpptraj currently recognizes the formats listed in 28.1, although it cannot write in all formats. In addition, a data set must be valid for the data file format. For example, 3D data (such as a grid) can be written to an OpenDX format file but not a Grace format file.

The default file format is called 'Standard', which simply has data in columns, like *ptraj*, although multiple data sets can be directed to the same output file. The format of a file can be changed either by specifying a recognized keyword (either on the command line itself or later via a 'datafile' command) or by giving the file an extension corresponding to te format, so 'filename.agr' will output in Grace format, and 'filename.gnu' will output in Gnuplot contour, and so on. The xmgrace/gnuplot output is particularly nice for the secstruct sumout and rmsd perresout files. Additional options for data files can be found in 28.4 on page 561.

Any action using the “out” keyword will allow data sets from separate commands to be written into the same file. For example, the commands:

```
dihedral phi :1@C :2@N :2@CA :2@C out phipsi.dat
dihedral psi :2@N :2@CA :2@C :3@N out phipsi.dat
```

will assign the “phi” and “psi” data sets generated from each action to the standard data output file “phipsi.dat”:

```
#Frame   phi   psi
```

Format	Filename Extensions	Keyword	Valid Dimensions	Notes
Standard	.dat	dat	1D, 2D, 3D	
Grace	.agr, .xmgr	grace	1D	
Gnuplot	.gnu	gnu	1D, 2D	
Xplor	.xplor, .grid	xplor	3D	
OpenDX	.dx	opendx	3D	
Amber REM log	.log	remlog	-	Read Only
Amber MDOUT	.mdout	mdout	-	Read Only
Evecs	.evecs	evecs	Modes data set only	
Vector pseudo-traj	.vectraj	vectraj	Vector data set only.	Write Only.
Gromacs XVG	.xvg	xvg	-	Read Only
CCP4	.ccp4	ccp4	3D	

Table 28.1.: *DataFile formats recognized by cpptraj. 'Valid Dimensions' shows what dimensions the format is valid for (e.g. you cannot write a 1D data set with OpenDX format).*

28.3.1. Data Set Selection Syntax

Many analysis commands can be used to analyze multiple data sets. The general format for selecting data sets is:

```
<name>[<aspect>]:<index>
```

The '*' character can be used as a wild-card for *entire* names (no partial matches).

- **<name>:** The data set name, usually specified in the action (e.g. in 'distance d0 @1 @2' the data set name is "d0").
- **<aspect>:** Optional; this is set for certain data sets internally in order to easily select subsets of data. **The brackets are required**. For example, when using 'hbond series', both solute-solute and solute-solvent hydrogen bond time series may be generated. To select all solute-solute hydrogen bonds one would use the aspect "[solutehb]"; to select solute-solvent hydrogen bonds the aspect "[solventhb]" would be used. Aspects are hard-coded and are listed in the commands that use them.
- **<index>:** Optional; for actions that generate many data sets (such as 'rmsd perres') an index is used. Depending on the action, the index may correspond to atom #s, residue #s, etc. A number range (comma and/or dash separated) may be used.

For example: to select all data sets with aspect "[shear]" named NA_00000:

```
NA_00000[shear]
```

To select all data sets with aspect "[stagger]" with any name, indices 1 and 3:

```
*[stagger]:1,3
```

In ensemble mode, data set selection has additional syntax:

```
<name>[<aspect>]:<index>%<member>
```

Where <member> is the ensemble member number starting from 0.

28.3.2. Data Set Math

As of version 15, *cpptraj* can perform basic math operations, even on data sets (with some limitations). Currently recognized operations are:

Operation	Symbol
Minus	-
Plus	+
Divide	/
Multiply	*
Power	^
Negate	-
Assign	=

Several functions are also supported:

Function	Form
Square Root	sqrt()
Exponential	exp()
Natural Logarithm	ln()
Absolute Value	abs()
Sine	sin()
Cosine	cos()
Tangent	tan()
Summation	sum()
Average	avg()
Standard Deviation	stdev()
Minimum	min()
Maximum	max()

Numbers can be expressed in scientific notation using "E" notation, e.g. 1E-5 = 0.00001. The parser also recognizes PI as the number pi. Expressions can also be enclosed in parentheses. So for example, the following expression is valid:

```
> 1 - ln(sin(PI/4) * 2)^2
Result: 0.879887
```

Results of numerical calculations like the above can be assigned to a variable (essentially a data set of size 1) for use in subsequent calculations, e.g.

```
> R = 1 - ln(sin(PI/4) * 2)^2
Result stored in 'R'
> R + 1 Result: 1.879887
```

Data sets can be specified in expressions as well. Currently data sets in an expression must be of the same type and only 1D, 2D, and 3D data sets are supported. Functions are applied to each member of the data set. So for example, given two 1D data sets of the same size named D0 and D1, the following expression:

```
> D2 = sqrt( D0 ) + D1
```

would take the square root of each member of D0, add it to the corresponding member of D1, and assign the result to D2. The following table lists which operations are valid for data set types. If a type is not listed it is not supported:

Data Set Type	Supported Ops	Supported Funcs	Notes
1D (integer, double, float)	All	All	
1D (vector)	+, -, *, /, =	None	'*' is dot product
2D (matrices)	+, -, /, *, =	sum, avg, stdev, min, max	
3D (grids)	+, -, /, *, =	sum, avg, stdev, min, max	

28.4. Data File Options

There is a subsection of keywords that can be used to modify data files which have been declared with an **'out'** keyword, create new data files from declared data sets, or when reading in data sets from files. Note that write-related keywords can be specified to most actions that use the **'out'** keyword directly without using the ***datafile*** command. For example, the **'time'** argument can be passed directly to the output from a ***distance*** command:

```
distance d0 :1 :2 out d0.agr time 0.001
```

The data file format can be changed from standard implicitly by using specific filename extensions or keywords. If the extension is not recognized or no keyword is give the default format is 'Standard'. Keywords and extensions for data file formats recognized by *cpptraj* are shown in 28.1. Note that the use of certain options may be restricted for certain data file formats.

```
[<format keyword>]
[{xlabel|ylabel|zlabel} <label>] [{xmin|ymin|zmin} <min>]
[{xstep|ystep|zstep} <step>] [time <dt>] [prec <width>[.<precision>]]
```

{xlabel | ylabel | zlabel} <label> Set the x-axis label for the specified datafile to <label>. For regular data files this is the header for the first column of data. If the data is at least 2-dimensional 'datafile ylabel <label>' will likewise set the y-axis label.

{xmin | ymin | zmin} <min> Set the starting X coordinate value to <min>. If the data is at least 2-dimensional 'datafile ymin <min>' will likewise set the starting Y coordinate value.

{xstep | ystep | zstep} <step> Multiply each frame number by <step> (x coordinates). If the data is at least 2-dimensional 'datafile ystep <step>' will likewise multiply y coordinates by <step>.

time <dt> Equivalent to the *ptraj* argument 'time' that could be specified with many actions. Multiplies frame numbers (x-axis) by <dt>.

prec <width>[.<precision>] Change the output format width (and optionally precision) of all sets *subsequently* added to the data file (i.e. does not change the precision of any data sets currently in the file). For example,

```
prec 12.4
prec 10
```

28.4.1. Standard Data File Options

Write

```
[invert] [noxcol] [noheader] [square2d] [nosquare2d]
```

invert Normally, data is written out with X-values pertaining to frames (i.e. data over all trajectories is printed in columns). This command flips that behavior so that X-values pertain to data sets (i.e. data over all trajectories is printed in rows).

noxcol Prevent printing of indices (i.e. the #Frame column in most datafiles) for the specified datafile. Useful e.g. if one would like a 2D plot such as phi vs psi. For example, given the input:

```
dihedral phi :1@C :2@N :2@CA :2@C out phipsi.dat
dihedral psi :2@N :2@CA :2@C :3@N out phipsi.dat
datafile phipsi.dat noxcol
```

Cpptraj `will write a 2 column datafile containing only phi and psi, no frame numbers will be written.`

noheader `Prevent printing of header line (e.g. '#Frame    D1') at the beginning of data file.`

square2d `Write 2D data as a square matrix, e.g.:`

```
<1,1> <2,1> <3,1>
<1,2> <2,2> <3,2>
```

nosquare2d `Write 2D data in 3 columns as:`

```
<X> <Y> <Value>
```

Read

```
[index <col>] [read2d] [vector] [mat3x3]
```

index <col> `Use column <col> (starting from 1) as index column (1D data only).`

read1d `Read data as 1D data sets (default).`

read2d `Read data as 2D square matrix.`

vector `Read data as vector. If indices are present they will be skipped. Assume first 3 columns after the index volumn are vector X, Y, and Z, and (if present) the next 3 columns contain vector origin X, Y, and Z.`

mat3x3 `Read data as 3x3 matrix. If indices are present they will be skipped. Assume matrices are in row major order on each line, i.e. M(1,1) M(1,2) ... M(3,2) M(3,3).`

By default, standard data files are assumed to contain 1D data in columns. Data set legends will be read in if the file has a header line (denoted by '#'). Columns labeled '#Frame' are automatically considered the 'index' column and skipped. Data sets are stored as <name>:<idx> where <name> is the given data set name (the file name if not specified) and <idx> corresponds to the column the data was read from starting from 1. *Cpptraj* assumes the data increases monotonically and will automatically attempt to determine the dimensions of the data set(s); a warning will be printed if this is not successful.

28.4.2. Grace Data File Options

For more information on Grace see http://plasma-gate.weizmann.ac.il/Grace/.

Write

```
[invert]
```

invert `Normally, data is written out with X-values pertaining to frames (i.e. data over all trajectories is printed in columns). This command flips that behavior so that X-values pertain to data sets.`

Read

Cpptraj will read set legends from grace files, and data sets are stored as <name>:<idx> where <name> is the given data set name (the file name if not specified) and <idx> corresponds to the set number the data was read from starting from 0.

28.4.3. Gnuplot Data File Options

For more information on these options it helps to look at the PM3D options in the Gnuplot manual (see http://www.gnuplot.info/).

Write

```
[nolabels] [usemap] [pm3d] [nopm3d] [jpeg] [noheader]
[{xlabels|ylabels|zlabels} <labellist>]
```

nolabels `Do not print axis labels.`

usemap `pm3d output with 1 extra empty row/col (may improve look).`

pm3d `Normal pm3d map output.`

nopm3d `Turn off pm3d`

jpeg `Plot will write to a JPEG file when used with gnuplot.`

noheader `Do not format plot; data output only.`

palette <arg> `Change gnuplot pm3d palette to <arg>:`

- **'rgb'** `Red, yellow, green, cyan, blue, magenta, red.`
- **'kbvyw'** `Black, blue, violet, yellow, white.`
- **'bgyr'** `Blue, green, yellow, red.`
- **'gray'** `Grayscale.`

xlabels|ylabels|zlabels <labellist> `Set x, y, or z axis labels with comma-separated list, e.g. 'xlabels X1,X2,X3'.`

28.4.4. Amber REM Log Options

Note that multiple REM logs can be specified in a single ***readdata*** command. See 28.10.25 on page 671 for more on replica log analysis.

Read

```
[nosearch] [dimfile <file>] [crdidx <crd indices>]
```

[nosearch] `If specified do not automatically search for MREMD dimension logs.`

[dimfile <file>] `remd.dim file for processing MREMD logs.`

[crdidx <crd indices>] `Use comma-separated list of indices as the initial coordinate indices (H-REMD only). For example (4 replicas):`

```
crdidx 4,2,3,1
```

28.4.5. Amber MDOUT Options

Note that multiple MDOUT files can be specified in a single ***readdata*** command.

28.4.6. Evecs File Options

Read

```
[ibeg <firstmode>] [iend <lastmode>]
```

ibeg <firstmode> `Number of the first mode (or principal component) to read from evecs file. Default 1.`

iend <lastmode> `Number of the last mode (or principal component) to read from evecs file. Default is to read all for newer evecs files (generated by` *`cpptraj`* `version > 12), 50 for older evecs files.`

28.4.7. Vector psuedo-traj Options

This can be used to write out a representation of a vector data set which can then be visualized. See 28.9.79 on page 645 for more on generating vector data sets.

Write

```
[trajfmt <format>] [parmout <file>] [noorigin]
```

trajfmt <format> `Output pseudo-trajectory format. See 28.8 on page 581 for trajectory format keywords.`

parmout <file> `File to write pseudo-trajectory topology to.`

[noorigin] `Do not write vector origin coordinates.`

28.4.8. OpenDX file options

Write

```
[bincenter] [gridwrap] [gridext]
```

bincenter `Center grid points on bin centers instead of corners.`

gridwrap `Like 'bincenter', but also wrap grid density. Useful when grid encompasses unit cell.`

gridext `Like 'bincenter', but also print extra layer of empty bins.`

28.4.9. CCP4 file options

Write

```
[title <title>]
```

[title <title>] `Set CCP4 output title.`

28.5. Coordinates (COORDS) Data Set Commands

Coordinate I/O tends to be the most time-consuming part of trajectory analysis. In addition, many types of analyses (for example two-dimensional RMSD and cluster analysis) require using coordinate frames multiple times. To simplify this, trajectory coordinates may be saved as a separate data set via the ***loadcrd*** command or ***createcrd*** action. Any action can then be performed on the COORDS data set with the ***crdaction*** command. The ***crdout*** command can be used to write coordinates to an output trajectory (similar to ***trajout***).

Although COORDS data sets store everything internally with single-precision, they can still use a large amount of memory. Because of this there is a specialized type of COORDS data set called a TRAJ data set (trajectory), which functions exactly like a COORDS data set except all data is stored on disk. TRAJ data sets can be created with the ***loadtraj*** command. *TRAJ data sets cannot be modified.*

There are several analyses that can be performed using COORDS data sets, either as part of the normal analysis list or via the ***runanalysis*** command. Note that while these analyses can be run on specified COORDS data sets, if one is not specified a default COORDS data set will be created, made up of frames from ***trajin*** commands.

As an example of where this might be useful is in the calculation of atomic positional fluctuations. Previously this required two steps: one to generate an average structure, then a second to rms-fit to that average structure prior to calculating the fluctuations. This can now be done in one pass with the following input:

```
parm topology.parm7
loadcrd mdcrd.nc
# Generate average structure PDB, @CA only
```

```
crdaction mdcrd.nc average avg.pdb @CA
# Load average structure PDB as reference
parm avg.pdb
reference avg.pdb parm avg.pdb
# RMS-fit to average structure PDB
crdaction mdcrd.nc rms reference @CA
# Calculate atomic fluctuations for @CA only
crdaction mdcrd.nc atomicfluct out fluct.dat bfactor @CA
```

The following COORDS data set commands are available:

Command	Description
combinecrd	Combine two or more COORDS sets.
crdaction	Run a single Action on a COORDS set.
crdout	Write a COORDS set to a file.
createcrd	(Action) Create a COORDS set during a Run.
loadcrd	Create or append to a COORDS set from a file.
loadtraj	Create special COORDS set where frames remain on disk.
permutedihedrals	Rotate specified dihedral(s) in given COORDS set by specific interval or to random values.
reference	Load a single trajectory frame as a reference.
rotatedihedral	Rotate specified dihedral to specified value or by given increment.

28.5.1. combinecrd

```
combinecrd <crd1> <crd2> ... [parmname <topname>] [crdname <crdname>]
```

<crdX> `COORDS data set to combine, specify 2 or more.`

[parmname <topname>] `Name of combined Topology.`

[crdname <crdname>] `Name of combined COORDS data set.`

Combined two or more COORDS data sets into a single COORDS data set. Note that the resulting topology will most likely **not** be usable for MD simulations. Box information will be retained - the largest box dimensions will be used.

For example, to load two MOL2 files as COORDS data sets, combine them, and write them out as a single MOL2:

```
loadcrd Tyr.mol2 CRD1
loadcrd Pry.mol2 CRD2
combinedcrd CRD1 CRD2 parmname Parm-1-2 crdname CRD-1-2
crdout CRD-1-2 Tyr.Pry.mol2
```

28.5.2. crdaction

```
crdaction <crd set> <actioncmd> [<action args>] [crdframes <start>,<stop>,<offset>]
```

Perform action <actioncmd> on COORDS data set <crd set>. A subset of frames in the COORDS data set can be specified with ’crdframes’.

For example, to calculate RMSD for a previously created COORDS data set named crd1 using frames 1 to the last, skipping every 10:

```
crdaction crd1 rmsd first @CA out rmsd-ca.agr crdframes 1,last,10
```

28.5.3. createcrd

This command is actually an Action that can be used to create COORDS data sets during trajectory processing, see 28.9.18 on page 598.

28.5.4. crdout

```
crdout <crd set> <filename> [<trajout args>] [crdframes <start>,<stop>,<offset>]
```

Write COORDS data set <crd set> to trajectory named <filename>. A subset of frames in the COORDS data set can be specified with 'crdframes'.

For example, to write frames 1 to 10 from a previously created COORDS data set named "crd1" to separate PDB files:

```
crdout crd1 crd1.pdb multi crdframes 1,10
```

28.5.5. loadcrd

```
loadcrd <filename> [parm <parm> | parmindex<#>] [<trajin args>] [name <name>]
```

Immediately load trajectory <filename> as a COORDS data set named <name> (default base name of <filename>). If <name> is already present the coordinates will be appended to the existing data set.

For example, to load frames from trajectories named 'traj1.nc' and 'traj2.nc' into a COORDS data set named Crd1:

```
loadcrd traj1.nc name Crd1
loadcrd traj2.nc name Crd2
```

28.5.6. loadtraj

```
loadtraj name <setname> [<filename>]
```

This command functions in two ways. If <filename> is not provided, all currently loaded input trajectories (from ***trajin*** commands) are added to TRAJ data set named <setname>. **Note that if the input trajectory list is cleared (via 'clear trajin') this will invalidate the TRAJ data set.** In addition, currently all trajectories must have the same number of atoms. Otherwise add trajectory <filename> to TRAJ data set <setname>.

TRAJ data sets cannot be modifed.

28.5.7. permutedihedrals

```
permutedihedrals crdset <COORDS set> resrange <range> [{interval | random}]
              [outtraj <filename> [<outfmt>]] [crdout <output COORDS>] [<dihedral types>]
  Options for 'random':
[rseed <rseed>] [out <# problems file> [<set name>]]
[ check [cutoff <cutoff>] [rescutoff <rescutoff>] [checkallresidues]
  [backtrack <backtrack>] [increment <increment>] [maxfactor <max_factor>] ]
  Options for 'interval':
<interval deg>
  <dihedral types> =  alpha beta gamma delta epsilon zeta nu1 nu2 h1p c2p chin phi psi chip omega
```

crdset <COORDS set> `COORDS data set to operate on.`

resrange <range> `Residue range to search for dihedrals.`

interval `Rotate found dihedrals by <interval>. This is done in an ordered fashion so that every combination of dihedral rotations is sampled at least once.`

random `Rotate each found dihedral randomly.`

[outtraj <filename>] `Trajectory file to write coordinates to.`

- **[<outfmt>]** `Trajectory file format.`

[crdout <output COORDS>] `COORDS data set to write coordinates to.`

<dihedral type> `One or more dihedral types to search for.`

`Options for 'interval:`

<interval deg> `Amount to rotate dihedral by each step.`

`Options for 'random':`

[rseed <rseed>] `Random number seed.`

[out <# problems file>] `File to write number of problems (clashes) each frame to.`

[<set name>] `Number of problems data set name.`

[check] `Check randomly rotated structure for clashes.`

- **[cutoff <cutoff>]** `Atom cutoff for checking for clashes (default 0.8 Å).`
- **[rescutoff <cutoff>]** `Residue cutoff for checking for clashes (defualt 10.0 Å).`
- **[checkallresidues]** `If specified all residues checked for clashes, otherwise only residues up to the currently rotated dihedral check.`
- **[backtrack <backtrack>]** `If a clash is encountered at dihedral N and cannot be resolved, go to dihedral N-<backtrack> to try and resolve the clash (default 4).`
- **[increment <increment>]** `If a clash is encountered, first attempt to rotate dihedral by increment to resolve it; if it cannot be resolved by a full rotation the calculation will backtrack (default 1).`
- **[maxfactor <max_factor>]** `The maximum number of total attempted rotations will be <max_factor> * <total # of dihedrals> (default 2).`

Create a trajectory by rotating specified dihedrals in a structure by regular intervals (**interval**), or create 1 structure by randomly rotating specified dihedrals (**random**). When randomly rotating dihedrals steric clashes will be checked if **check** is specified; in such cases the algorithm will attempt to resolve the clash as best it can. If clashes are not being resolved you can increase the number of rotation attempts *cpptraj* will make by increasing **maxfactor**.

For example, to rotate all backbone dihedrals in a protein with coordinates in a file named tz2.rst7 in -120 degree intervals and write the resulting trajectory in Amber format to rotations.mdcrd:

```
reference tz2.rst7 [TZ2]
permutedihedrals crdset [TZ2] interval -120 outtraj rotations.mdcrd phi psi
```

To randomly rotate backbone dihedrals for the same structure and write to file random.mol2 in MOL2 format:

```
reference tz2.rst7 [TZ2]
permutedihedrals crdset [TZ2] random rseed 1 check maxfactor 10 phi psi \
                  outtraj random.mol2 multi
```

28.5.8. reference

Reference coordinates can now be used and manipulated like COORDS data sets. See 28.8.3 on page 582 for command syntax.

28.5.9. rotatedihedral

```
rotatedihedral crdset <COORDS set> [frame <#>] [name <output set name>]
               {value <value> | increment <increment>}
               { <mask1> <mask2> <mask3> <mask4> |
                 res <#> type <dih type> }
<dih type> = alpha beta gamma delta epsilon zeta nu1 nu2 h1p c2p chin phi psi chip omega
```

crdset <COORDS set> `Coordinates data set to work on. If a TRAJ data set is specified, name must also be specified.`

[frame <#>] `Frame of the COORDS set to work on.`

[name <output set name>] `Output COORDS set. If not specified the input COORDS set will be modified.`

value <value> `Set specified dihedral to given value in degrees.`

increment <increment> `Increment specified dihedral by increment in degrees.`

<mask1> <mask2> <mask3> <mask4> `Define dihedral by atom masks. Each mask should only select one atom.`

res <#> `Rotate dihedral specified by type in residue number <#>.`

type <dih type> `Dihedral type to rotate in specified residue.`

Rotate the specified dihedral in given COORDS set to a target value or by given increment. For example, to set the protein chi dihedral in residue 8 to 35 degrees and write out to a mol2 file:

```
parm ../tz2.parm7
loadcrd ../tz2.nc 1 1 name TZ2
rotatedihedral crdset TZ2 value 35 res 8 type chip
crdout TZ2 tz2.rotate.1.mol2
```

28.6. General Commands

The following general commands are available:

Command	Description
activeref	Select the reference for distance-based masks.
clear	Clear various objects from the cpptraj state.
create	Create (but do not yet write) a data file.
datafile	Used to manipulate data files.
datafilter	Filter data sets based on given criteria.
dataset	Use to manipulate data sets.
debug | prnlev	Set debug level. Higher levels give more info.
exit | quit	Quit cpptraj.
go | run	Start a trajectory processing Run.
help	Provide help for commands.
list	List various objects in the cpptraj state.
noexitonerror	Attempt to continue even if errors are encountered.
noprogress	Do not print a progress bar during a Run.
precision	Change the output precision of data sets.
printdata	Print data set to screen.
readdata	Read data sets from files.
readinput	Read cpptraj input from a file.
removedata	Remove specified data set(s).
rst	Generate Amber-style distance/angle/torsion restraints.
runanalysis	Run an analysis immediately or run all queued analyses.
select	Print the results of an atom mask expression.
selectds	Print the results of a data set selection expression.
silenceactions	Prevent Actions from writing information to STDOUT.
write | writedata	Immediately write data to a file or write to all current data files.

28.6.1. activeref

```
activeref <#>
```

Set which reference structure should be used when setting up distance-based masks for everything but the ’mask’ action. Numbering starts from 0, so ’activeref 0’ selects the first reference structure read in, ’activeref 1’ selects the second, and so on.

28.6.2. clear

```
clear [{all | <type>}]
      (<type> = actions,trajin,trajout,ref,parm,analysis,datafile,dataset)
```

Clear list of indicated type, or all lists if ’all’ specified. Note that when clearing actions or analyses, associated data sets and data files are not cleared and vice versa.

28.6.3. create

```
create <filename> <datasetname0> [<datasetname1> ...] [<DataFile Options>]
```

Add specified data sets to the data file named <filename>; if the file does not exist, it will be added to the DataFileList. Data files created in this way are only written at the end of coordinate processing, analyses, or via the ***’writedata’*** command. See 28.4 on page 561 for more data file format options.

28.6.4. datafile

```
datafile <filename> <datafile arg>
```

Pass <datafile arg> to data file <filename>. See 28.4 on page 561 for more details.

28.6.5. datafilter

```
datafilter <dataset arg> min <min> max <max> [out <file> [name <setname>]]
```

<dataset arg> min <min> max <max> `Data set and min/max cutoffs to use; can specify more than once.`

[out <file>] `Write out to file named <file>.`

[name <setname>] `Name of filter data set.`

Create a data set (optionally named <setname>) containing 1 for data within given **<min>** and **<max>** criteria for each specified data set. There must be at least one **<min>** and **<max>** argument, and can be as many as there are specified data sets. For example, to read in data from two separate files (d1.dat and a1.dat) and generate a filter data set named FILTER having 1 when d1 is between 0.0 and 3.0 and a1 is between 135.0 and 180.0:

```
readdata a1.dat name a1
readdata d1.dat name d1
datafilter d1 min 0.0 max 3.0 a1 min 135.0 max 180.0 out filter.dat name FILTER
```

Note that a similar command that can be used with data generated by Actions during trajectory processing is ***filter*** (see page 605).

28.6.6. dataset

```
dataset { legend <legend> <set> |
          makexy <Xset> <Yset> [name <name>] |
       cat <set0> <set1> ... [name <name>] [nooffset] |
          make2d <1D set> cols <ncols> rows <nrows> [name <name>] |
          remove <criterion> <select> <value> [and <value2>] [<set selection>] |
       [mode <mode>] [type <type>] <set arg1> [<set arg 2> ...] }
  <mode>:  distance angle torsion pucker rms
  <type>:  alpha beta gamma delta epsilon zeta pucker chi h1p c2p
           phi psi pchi omega noe
    Options for 'type noe':
       [bound <lower> bound <upper>] [rexp <expected>] [noe_strong] [noe_medium] [noe_weak]
```

legend <legend> <set> `Set the legend for data set <set>.`

[name <name>] `New data set name for makexy/cat/make2d.`

makexy <Xset> <Yset> `Create a new data set (optionally named <name>) with X values from <Xset> and Y values from <Yset>.`

cat <set0> <set1> ... `Concatenate two or more data sets into a new data set (optionally named <name>).`

make2d <1D set> cols <ncols> rows <nrows> `Convert 1D data set into row-major 2D data set with specified number of rows and columns.`

remove <criterion> <select> <value> [and <value2>] [<set selection>] `Remove data sets from <set selection> according to specified criterion and selection.`

```
<criterion>:  'ifaverage' 'ifsize' 'ifmode' 'iftype'
<select>   :  'equal' '==' 'notequal' '!=' 'lessthan' '<' 'greaterthan' '>' 'between' 'outside'
```

[mode <mode>] `Set data set(s) mode to <mode>.`

[type <type>] `Set data set(s) type to 'type', useful for e.g. analysis with` ***`statistics`***`. Note this can also be done with 'type <type>' for certain commands (`***`distance`***`,` ***`dihedral`***`,` ***`pucker`*** `etc). Note that not every <type> is compatible with a given <mode>.`

`Options for type noe only:`

[bound <lower> bound <upper>] `Lower and upper bounds for NOE (in Angstroms); must specify both.`

[rexp <expected>] `Expected value for NOE (in Angstroms); if not given '(<lower> + <upper>)' / 2.0 is used.`

[noe_strong] `Set lower and upper bounds to 1.8 and 2.9 Å respectively.`

[noe_medium] `Set lower and upper bounds to 2.9 and 3.5 Å respectively.`

[noe_weak] `Set lower and upper bounds to 3.5 and 5.0 Å respectively.`

Either set the legend for a single data set, create a new set with X values from one set and Y values from another, concatenate 2 or more sets, make a 2D set from 1D set, remove sets according to a certain criterion, or change the mode/type for one or more data sets.

Setting the mode/type can be useful for cases where the data set is being read in from a file; for example when reading in a dihedral data set the type can be set to 'dihedral' so that various Analysis routines like ***statistics*** know to treat it as periodic. A brief description of possible modes and types follows:

Mode	Type	Description
distance	noe	NOE distance.
angle		Angle.
torsion	alpha	Nucleic acid alpha.
	beta	Nucleic acid beta.
	gamma	Nucleic acid gamma.
	delta	Nucleic acid delta.
	epsilon	Nucleic acid epsilon.
	zeta	Nucleic acid zeta.
	nu1	Nucleic pucker (O4').
	nu2	Nucleic pucker (C4').
	h1p	Nucleic acid H1'.
	c2p	Nucleic acid C2'.
	chin	Nucleic acid chi.
	phi	Protein Phi.
	psi	Protein psi.
	chip	Protein chi.
	omega	Protein omega.
pucker	pucker	Sugar pucker.
rms		RMSD.
matrix	distance	Distance matrix.
	covariance	Cartesian covariance matrix.
	'mass-weighted covariance'	Mass weighted Cartesian covariance matrix.
	correlation	Dynamic cross correlation matrix.
	'distance covariance'	Distance covariance matrix.
	IDEA	IDEA matrix.
	IRED	IRED matrix.
	'dihedral covariance'	Dihedral covariance matrix.
vector	IRED	IRED vector.

28.6.7. debug | prnlev

```
debug [<type>] <#>
      (<type> = actions,trajin,trajout,ref,parm,analysis,datafile,dataset)
```

Set the level of debug information to print. In general the higher the <#> the more information that is printed. If <type> is specified only set the debug level for a specific area of *cpptraj*.

28.6.8. exit | quit

Exit normally.

28.6.9. go | run

Begin trajectory processing, followed by analysis and datafile write.

28.6.10. help

```
help {[<command>] | General | Action | Analysis | Topology | Trajectory}
```

By itself, list all commands known to *cpptraj*. If given with a command, print help for that command. Otherwise, list all commands of a certain category (General, Action, Analysis, Topology, or Trajectory).

28.6.11. list

```
list <type>
     (<type> = actions,trajin,trajout,ref,parm,analysis,datafile,dataset)
```

List the currently loaded objects of <type>.

28.6.12. noexitonerror

```
noexitonerror
```

Normally *cpptraj* will exit if actions fail to initialize properly. If ***noexitonerror*** is specified, *cpptraj* will attempt to continue past such errors. This is the default if in interactive mode.

28.6.13. noprogress

```
noprogress
```

Do not display read progress during trajectory processing.

28.6.14. precision

```
precision {<filename> | <dataset arg>} [<width>] [<precision>]
```

Set the precision for all data sets in data file <filename> or data set(s) specified by <dataset arg> to *width.precision*, where width is the column width and precision is the number of digits after the decimal point. Note that the <precision> argument only applies to floating-point data sets.

For example, if one wanted to set the precision of the output of an Rmsd calculation to 8.3, the input could be:

```
trajin ../run0.nc
rms first :10-260 out prec.dat
precision prec.dat 8 3
```

and the output would look like:

```
#Frame RMSD_00000
1 0.000
2 0.630
```

28.6.15. readdata

```
readdata <filename>  [name <dsname>] [as <fmt>] [<format options>]
```

name <dsname> `Name for read-in data set(s).  Default is <filename>.`

as <fmt> `Force <filename> to be read as a specific format using given format keyword.`

Read data from file <filename> and store as data sets. For more information on formats currently recognized by cpptraj see 28.1 on page 559. For format-specific options see 28.4. For example, given the file calc.dat:

```
#Frame  R0   D1
1      1.7 2.22
```

The command '`readdata calc.dat`' would read data into two data sets, calc.dat:2 (legend set to "R0") and calc.dat:3 (legend set to "D1").

28.6.16. readinput

```
readinput <filename>
```

Read *cpptraj* commands from file <filename>.

28.6.17. removedata

```
removedata <arg>
```

Remove data set corresponding to <arg>.

28.6.18. rst

```
rst <mask1> <mask2> [<mask3>] [<mask4>]
    r1 <r1> r2 <r2> r3 <r3> r4 <r4> rk2 <rk2> rk3 <rk3>
    {[parm <parmfile / tag> | parmindex <#>]}
    [{ref <refname> | refindex <#> | reference} [offset <off>] [width <width>]]
    [out <outfile>]
```

<mask1> `(Required) First atom mask.`

<mask2> `(Required) Second atom mask.  If only two masks assume distance restraint.`

[<mask3>] `(Optional) Third atom mask.  If 3 atom masks assume angle restraint.`

[<mask4>] `(Optional) Fourth atom mask.  If 4 atom masks assume dihedral restraint.`

rX <rX> `Value of RX (X=1-4, default 0.0)`

rk2 <rk2> `Value of RK2 (force constant to be applied when R is R1 <= R < R2)`

rk3 <rk3> `Value of RK3 (force constant to be applied when R is R3 <= R < R4)`

[parm <parmfile / tag> | parmindex <#>] `Topology to be used for atom masks.`

{ref <refname> | refindex <#> | reference} `Use distance/angle/dihedral in reference structure to determine values for r1, r2, r3, and r4.  The value of r2 is set to <r2> + <off>, r3 = r2, r1 = r2 - <width>, r4 = r3 + <width>.`

[offset <off>] `(Reference only) Value to offset distance/angle/torsion in reference by (default 0.0).`

[width <width>] `(Reference only) Width between r1 and r2, r3 and r4 (default 0.5).`

[out <outfile>] `Write restraints to outfile.  If not specified, write to STDOUT.`

Generate Amber-style distance restraints for use with nmropt=1. This is particularly useful for generating distance restraints based off of reference coordinates. For example to generate a distance restraint between two C5' atoms using the current distance between them in a reference structure, offsetting the distance by 1.0 Ang.:

```
parm 30bp-longbox-tip3p-na.parm7
reference 30bp-longbox.rst7
rst :1@C5' :31@C5' reference offset 1.0 rk2 10.0 rk3 10.0 out output
```

28.6.19. runanalysis

```
runanalysis [<analysiscmd> [<analysis args>]]
```

Run given analysis command immediately and write any data generated. If no command is given run any analysis currently set up. NOTE: When 'runanalysis' is specified alone, data is not automatically written; to write data generated with 'runanalysis' use the 'writedata' command (this allows multiple analysis runs between output if desired).

28.6.20. select

```
select <mask>
```

Prints the number of selected atoms corresponding to the given mask, as well as the atom numbers with format:

```
Selected= <#atom1> <#atom2> ...
```

This does not affect the state in any way, but is intended for use in scripts etc. for testing the results of a mask expression.

28.6.21. selectds

```
selectds <dataset arg>
```

Show the results of a data set selection. Data set selection has the format:

```
<name>[<aspect>]:<index>
```

Either the [<aspect>] or the <index> arguments may be omitted. A '*' can be used in place of <name> or [<aspect>] as a wildcard. The <index> argument can be a single number or a range separated by '-' and ','.

This command does not affect the state in any way, but is particularly useful in interactive mode for determining the results of a dataset argument.

28.6.22. write | writedata

```
write [<filename> <datasetname0> [<datasetname1> ...]] [<DataFile Options>]
```

With no arguments, write all files currently in the data file list. Otherwise, write specified data set(s) to <filename>. This is like the 'create' command except a data file is not added to the data file list; it is written immediately. See 28.4 on page 561 for more data file format options.

28.6.23. System Commands

These commands call the equivalent external system commands.

gnuplot <args> Call `gnuplot` (if it is installed on your system) with the given arguments.

head <args> Call `head`, which lists the first few lines of a file.

less <args> Call `less`, which can be used to view the contents of a file.

ls <args> List the contents of a directory.

pwd <args> Print the current working directory.

xmgrace <args> Call `xmgrace` (if it is installed on your system) with the given arguments.

28.7. Topology File Commands

These commands control the reading and writing of topology files. Cpptraj supports the following topology file formats:

Format	Keyword	Extension	Notes
Amber Topology	amber	.parm7	Only fully-supported format.
PDB	pdb	.pdb	Read Only
Mol2	mol2	.mol2	Read Only
CIF	cif	.cif	Read Only
Charmm PSF	psf	.psf	Limited Write
Gromacs Topology	gromacs	.top	Limited Write
SDF	sdf	.sdf	Read Only
Tinker ARC	arc	.arc	Read Only

For most commands that require a topology one can be specified via two keywords:

parm [<name>] Select topology corresponding to given file name, tag, or name.

parmindex [<#>] Select topology by order in which it was loaded, starting from 0.

The following topology related commands are available:

Command	Description
angleinfo, angles, printangles	Print angle info for selected atoms.
atominfo, atoms, printatoms	Print details for selected atoms.
bondinfo, bonds, printbonds	Print bond info for selected atoms.
charge	Print total charge for selected atoms.
comparetop	Compare two topologies and report differences.
dihedralinfo, dihedrals, printdihedrals	Print dihedral info for selected atoms.
mass	Print total mass for selected atoms.
molinfo	Print molecule info for selected atoms.
parm	Load a topology file.
parmbox	Modify box info for a loaded topology.
parminfo	Print details for selected topology.
parmstrip	Remove selected atoms from topology.
parmwrite	Write selected topology to file.
resinfo	Print residue info for selected atoms.
scaledihedralk	Scale selected dihedral force constants.
solvent	Change which molecules are considered solvent.

28.7.1. angleinfo | angles | printangles

```
angleinfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print angle information of atoms in <mask> for selected topology (first loaded topology by default) with format:

```
# Angle Kthet degrees atom names (numbers)
```

Where `Angle` is the internal angle index, `Kthet` is the angle force constant, `degrees` is the angle equilibrium value, `atom names` shows the atoms involved in the angle with format `:<residue num>@<atom name>`, and `(numbers)` shows the atom indices involved in a comma-separated list. Atom types will be shown in the last column.

28.7.2. atominfo | atoms | printatoms

```
atominfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print information on atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Atom Name #Res Name #Mol Type Charge Mass GBradius El [rVDW] [eVDW]
```

where `#Atom` is the internal atom index, the first `Name` column is the atom name, `#Res` is the atom's residue number, the second `Name` column is residue name, `#Mol` is the atom's molecule number, `Type` is the atom's type (certain topologies only), `Charge` is the atom charge (in units of electron charge), `Mass` is the atom's mass (in amu), `GBradius` is the generalized Born radius of the atom (Amber topologies only), and `El` is the 2 character element string. The final two columns are only shown if the topology contains non-bonded parameters: `rVDW` is the atom's Lennard-Jones radius and `eVDW` is the atom's Lennard-Jones epsilon.

28.7.3. bondinfo | bonds | printbonds

```
bondinfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print bond information for atoms in <mask> for selected topology (first loaded topology by default) with format:

```
# Bond Kb Req atom names (numbers)
```

where `Bond` is the internal bond index, `Kb` is the bond force constant, `Req` is the bond equilibrium value (in Angstroms), `atom names` shows both atom names with format `:<residue num>@<atom name>`, and `(numbers)` shows both atom numbers in a comma-separated list. Atom types will be shown in the last column.

28.7.4. charge

```
charge [parm <name> | parmindex <#> | <#>] <mask>
```

Print the total charge of atoms in <mask> (in units of electron charge) for selected topology (first loaded topology by default).

28.7.5. comparetop

```
comparetop {parm <name> | parmindex <#>} {parm <name> | parmindex <#>} [out <file>]
           [atype] [lj] [bnd] [ang] [dih] [atoms]
```

parm <name> | parmindex <#> `Topologies to compare.`

out <file> `Print results to file instead of screen.`

[atype] `Only report atom type differences.`

[lj] `Only report differences in Lennard-Jones parameters.`

[bnd] `Only report differences in bond parameters.`

[ang] `Only report differences in angle parameters.`

[dih] `Only report differences in dihedral parmeters.`

[atoms] `Only report differences in atom properties.`

Compare and report differences in atoms/parameters between two topologies. Differences are reported in standard 'diff' format, with '<' prefix indicating the parameter is from the first topology and '>' prefix indicating the parameter is from the second topology.

28.7.6. dihedralinfo | dihedrals | printdihedrals

```
dihedralinfo [parm <name> | parmindex <#> | <#>] [<mask>] [and]
```

Print dihedral information of atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Dihedral pk phase pn atoms
```

where `#Dihedral` is the internal dihedral index, `pk` is the dihedral force constant, `phase` is the dihedral phase, `pn` is the dihedral periodicity, and `atoms` shows the names of the atoms involved in the angle with format `:<residue num>@<atom name>`, followed by the atom indices involved in a comma-separated list. In addition if the dihedral is an end dihedral, improper dihedral, or both it will be prefaced with an `E`, `I`, or `B` respectively. Atom types will be shown in the last column.

If the **and** keyword is specified then printed dihedrals must contain all atoms specified in the <mask>. For example, to only get dihedrals containing the atom types N3, CT, and OH:

```
dihedrals @%N3,CT,OH and
```

28.7.7. mass

```
[<parmindex>] [parm <name> | parmindex <#> | <#>] <mask>
```

Print the total mass of atoms in <mask> (in amu) for selected topology (first loaded topology by default).

28.7.8. molinfo

```
molinfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print molecule information for atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Mol  Natom  #Res Name [SOLVENT]
```

where `#Mol` is the molecule number, `Natom` is the number of atoms in the molecule, and `#Res` and `Name` are the residue number and residue name of the first residue in the molecule respectively. `SOLVENT` will be printed if the molecule is currently considered a solvent molecule.

28.7.9. parm

```
parm <filename> [{[TAG] | name <setname>}] [nobondsearch | bondsearch [<offset>]]
```

<filename> `Parameter file to read in; format is auto-detected.`

'[TAG]' Optional tag (bounded in brackets) which can be referred to in place of the topology file name in order to simplify references to it (see 28.2.4 on page 557 for examples of how to use tags).

[name <setname>] Optional name that can be used to refer to the topology in place of the file name.

[bondsearch <offset>] Optional; when searching for bonds via geometry search (default for Topologies without bond information) add <offset> to distances (default 0.2 Å). Increase this if your system includes unusually long bonds.

[nobondsearch] Optional; if specified do not search for bonds via geometry if Topology does not include bond information. May cause some actions to fail.

Read in parameter file. The file format will be auto-detected. Current formats recognized by cpptraj are listed on page 575. If the file does not contain bond information, cpptraj will attempt to assign bonds based on a simple distance search of atoms within and between residues. The distance cutoff for determining bonds between atoms depends on the elements of the two atoms in question, augmented by <offset>. Molecule information is then determined from bond information.

28.7.9.1. *PDB format:*

```
[pqr] [readbox] [noconect]
```

[pqr]: Read charge and radius information from the occupancy and B-factor columns.

[readbox]: Read unit cell information from CRYST1 record if present.

[noconect] Do not read in CONECT records from PDB file.

IMPORTANT NOTES FOR PDB FILES

In some older format PDB files (before version 3), certain atoms may contain the asterisk ('*') character in their name (e.g. 'C1*' in a nucleic acid backbone). Since in *cpptraj* the asterisk is a reserved character for atom masks all asterisks in PDB atom names are replaced with single quote (') to avoid issues with the mask parser. So in a structure with an atom named C1*, to select it use the mask "@C1'".

Sometimes PDB files can contain alternate coordinates for the same atom in a residue, e.g.:

```
ATOM    806  CA ACYS A 105       6.460 -34.012 -21.801  0.49 32.23
ATOM    807  CB ACYS A 105       6.054 -33.502 -20.415  0.49 35.28
ATOM    808  CA BCYS A 105       6.468 -34.015 -21.815  0.51 32.42
ATOM    809  CB BCYS A 105       6.025 -33.499 -20.452  0.51 35.38
```

If this is the case *cpptraj* will print a warning about duplicate atom names but will take no other action. Both residues are considered 'CYS' and the mask ':CYS@CA' would select both atom 806 and 809. Residue insertion codes are read in but also not used by the mask parser.

28.7.10. parmbox

```
parmbox [parm <name> | parmindex <#> | <#>] [nobox] [truncoct]
     [x <xval>] [y <yval>] [z <zval>] [alpha <a>] [beta <b>] [gamma <g>]
```

[parm <name> | parmindex <#> | <#>] Name/tag or index of topology to modify. Default is first loaded topology.

[nobox] Remove box information.

[truncoct] `Set truncated octahedon angles with lengths equal to <xval>.`

[x <xval>] `Box X length.`

[y <yval>] `Box Y length.`

[z <zval>] `Box Z length.`

[alpha <a>] `Box alpha angle.`

[beta <b>] `Box beta angle.`

[gamma <g>] `Box gamma angle.`

Modify the box information for specified topology. Overwrites any box information if present with specified values; any that are not specified will remain unchanged. Note that unlike the ***box*** action this command affect box information immediately. This can be useful for e.g. removing box information from a parm when stripping solvent:

```
parm mol.water.parm7
parmstrip :WAT
parmbox nobox
parmwrite out strip.mol.nobox.parm7
```

28.7.11. parminfo

```
parminfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print a summary of information contained in the specified topology (first loaded topology by default) .

28.7.12. parmstrip

```
parmstrip <mask> [parm <name> | parmindex <#> | <#>]
```

Strip atoms in **<mask>** from specified topology (by default the first topology loaded). Note that unlike the ***strip*** Action, this permanently modifies the topology for as long as *cpptraj* is running, so this should not be used if the topology is being used to read or write a trajectory via ***trajin/trajout***. This command can be used to quickly created stripped Amber topology files. For example, to strip all residues name WAT from a topology and write a new topology:

```
parm mol.water.parm7
parmstrip :WAT
parmwrite out strip.mol.parm7
```

28.7.13. parmwrite

```
parmwrite out <filename> [{parm <name> | parmindex <#> | <#> | crdset <setname>}]
          [<fmt>] [nochamber]
```

<filename> `File to write to.`

[parm <name> | parmindex <#> | <#>] `Topology to write out.`

[crdset <setname>] `Write topology from specified COORDS data set.`

[<fmt>] `Format keyword. If not specified the file name extension will be used. Default is Amber Topology.`

[nochamber] `(Amber topology only) Remove any CHAMBER information from the topology.`

Write out specified topology (first topology loaded by default) to **<filename>** with format **<fmt>** (Amber topology if not specified). Note that the Amber topology format is the only fully supported format for topology writes.

28.7.14. resinfo

```
resinfo [parm <name> | parmindex <#> | <#>] [<mask>] [short]
```

Print residue information for atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Res  Name First  Last Natom #Orig #Mol
```

where `#Res` is the residue number, `Name` is the residue name, `First` and `Last` are the first and last atom numbers of the residue, `Natom` is the total number of atoms in the residue, `#Orig` is the original residue number (in PDB files), and #Mol is the molecule number.

If short is specified then residues will be printed out in a condensed format. Each residue name will be shortened to 1 character, and residues are printed out in groups of 10, 5 groups to a line, with each line beginning with a residue number, e.g.

```
> resinfo short 4
1         MGFLAGKKIL ITGLLSNKSI AYGIAKAMHR EGAELAFTYV GQFKDRVEKL
51        CAEFNPAAVL PCDVISDQEI KDLFVELGKV WDGLDAIVHS IAFAPRDQLE
```

If the 1 character name for a residue is unknown it will be shown as the first letter of the residue name in lower-case.

28.7.15. scaledihedralk

```
scaledihedralk [parm <name> | parmindex <#>] <scale factor> [<mask> [useall]]
```

Scale dihedral force constants for dihderals selected by <mask> for specified topology. If **useall** is specified all atoms in <mask> must be present to select a dihedral, otherwise any atom in <mask> will selected a dihedral.

28.7.16. solvent

```
solvent [parm <name> | parmindex <#> | <#>] { <mask> | none }
```

Set solvent for selected topology (first loaded topology by default) based on **<mask>**, or set nothing as solvent if **none** is specified.

28.8. Trajectory File Commands

These commands control the reading and writing of trajectory files. There are three trajectory types in *cpptraj*: input, output, and reference. In *cpptraj*, trajectories are always associated with a topology file. If a topology file is not specified, a trajectory file will be associated with the first topology file loaded by default (this is true for both input and output trajectories.

Cpptraj currently understands the following trajectory file formats:

Format	Keyword(s)	Extension	Notes
Amber Trajectory	crd	.crd	Default format if keywords/extensions not recognized.
Amber NetCDF	cdf, netcdf	.nc	No compression.
Amber Restart	restart	.rst7	
Amber NetCDF Restart	ncrestart, restartnc	.ncrst	
Charmm DCD	dcd, charmm	.dcd	
Charmm COR	cor	.cor	Read Only
PDB	pdb	.pdb	
Mol2	mol2	.mol2	
Scripps Binpos	binpos	.binpos	
Gromacs TRR	trr	.trr	
Gromacs GRO	gro	.gro	Read Only
Gromacs XTC	xtc	.xtc	
CIF	cif	.cif	Read Only
Tinker ARC	arc	.arc	Read Only
SQM Input	sqm	.sqm	Write Only
SDF	sdf	.sdf	Read Only
LMOD Conflib	conflib	.conflib	Read Only, Detection by extension

The following trajectory-related commands are available:

Command	Description
ensemble	Set up a trajectory ensemble for reading during a run.
ensemblesize	(MPI only) specify number of members expected in subsequent ***ensemble*** commands.
reference	Read in a reference structure.
trajin	Set up a trajectory for reading during a Run.
trajout	Set up an output trajectory or ensemble for writing during a Run.

28.8.1. ensemble

```
ensemble <file0> {[<start>] [<stop> | last] [offset]} | lastframe
         [parm <parmfile / tag> | parmindex <#>]
         [trajnames <file1>,<file2>,...,<fileN>
         [remlog <remlogfile> [nstlim <nstlim> ntwx <ntwx>]]
```

<file0> `Lowest replica filename.`

[<start>] `Frame to begin reading ensemble at (default 1).`

[<stop> | last] `Frame to stop reading ensemble at; if not specified or 'last' specified, end of trajectories.`

[<offset>] `Offset for reading in trajectory frames (default 1).`

[lastframe] `Select only the final frame of the trajectories.`

[parm <parmfile>] `Topology filename/tag to associate with trajectories (default first topology).`

[parmindex <#>] `Index of Topology to associate with trajectories (default 0, first topology).`

[trajnames <file1>,...,<fileN>] `Do not automatically search for additional replica trajectories; use comma-separated list of trajectory names.`

[remlog <remlogfile>] `For H-REMD trajectories only, use specified REMD log file to sort trajectories by coordinate index (instead of by Hamiltonian).`

[nstlim <nstlim> ntwx <ntwx>] `If trajectory and REMD log were not written at the same rate, these are the values for nstlim (steps between each exchange) and ntwx (steps between trajectory write) used in the REMD simulation.`

Read in and process trajectories as an ensemble. Similar to '***trajin*** remdtraj', except instead of processing one frame at a target temperature, process all frames. This means that action and trajout commands apply to the entire ensemble; note however that not all actions currently function in ***'ensemble'*** mode. For example, to read in a replica ensemble, convert it to temperature trajectories, and calculate a distance at each temperature:

```
parm ala2.99sb.mbondi2.parm7
ensemble rem.crd.000 trajnames rem.crd.001,rem.crd.002,rem.crd.003
trajout temp.crd
distance d1 out d1.ensemble.dat @1 @21
```

This will output 4 temperature trajectories named 'temp.crd.X', where X ranges from 0 to 3 with 0 corresponding to the lowest temperature, and 'd1.ensemble.dat' containing 4 columns, each corresponding to a temperature. If run with MPI, data will be written to separate files named 'd1.ensemble.dat.X', similar to the output trajectories.

Note that in parallel (i.e. MPI) users should specify the ***ensemblesize*** command prior to ***ensemble*** in order to improve set up efficiency.

28.8.2. ensemblesize

```
ensemblesize <#>
```

This command is MPI only. It is used to set the expected number of members in any subsequent ***ensemble*** command, which dramatically improves set up efficiency.

28.8.3. reference

```
reference <name> [<frame#>] [<mask>] ([tag]) [lastframe] [crdset]
          [parm <parmfile / tag> | parmindex <#>]
```

<name> `File name (or COORDS set name if 'crdset' specified) to read in as reference; any trajectory recognized by 'trajin' can be used.`

[<frame#>] `Frame number to use (default 1).`

[<mask>] `Only load atoms corresponding to <mask> from reference.`

([tag]) `Tag to give this reference file, e.g. "[MyRef]"; BRACKETS MUST BE INCLUDED.`

[lastframe] `Use last frame of reference.`

[crdset] `Use for COORDS data set named <name> instead of file.`

[parm <parmfile/tag>] `Topology filename/tag to associate with reference (default first topology).`

[parmindex <#>] `Index of Topology to associate with reference (default 0, first topology).`

Use specified trajectory as reference coordinates. For trajectories with multiple frames, the first frame is used if a specific frame is not specified. An optional tag can be given (bounded in brackets) which can then be used in place of the name (see 28.2.4 on page 557 for examples of how to use tags). If desired, an atom mask can be used to read in only specified atoms from a reference.

Reference coordinates are now considered COORDS data sets and can be used anywhere a COORDS data set could, which allows reference structures to be manipulated once they are loaded. For example, a reference structure could be centered on the origin like so:

```
reference tz2.rst7 [MyRef]
crdaction [MyRef] center origin
```

Note that the ’average’ keyword has been deprecated for reference. If desired, an averaged reference COORDS data set can be created from a trajectory using the ’average’ command like so:

```
parm myparm.parm7
trajin mytraj.nc
rms first :1-12
average crdset RefAvg
run
rms ToAvg reference :1-12 out ToAvg.dat
```

28.8.4. trajin

```
trajin <filename> {[<start> [<stop> | last] [<offset>]]} | lastframe
       [parm <parmfile / tag> | parmindex <#>]
       [mdvel <velocities>] [mdfrc <forces>]
       [ <Format Options> ]
       [ remdtraj {remdtrajtemp <Temperature> | remdtrajidx <#>}
                  [trajnames <file1>,<file2>,...,<fileN>] ]
```

<filename> `Trajectory file to read in.`

[<start>] `Frame to begin reading at (default 1).`

[<stop> | last] `Frame to stop reading at; if not specified or 'last' specified, end of trajectory.`

[<offset>] `Offset for reading in trajectory frames (default 1).`

[lastframe] `Select only the final frame of the trajectory.`

[parm <parmfile/tag>] `Topology filename/tag to associate with trajectory (default first topology).`

[parmindex <#>] `Index of Topology to associate with trajectory (default 0, first topology).`

[mdvel <velocities>] `Use velocities from specified file.`

[mdfrc <forces>] `Use forces from specified file.`

[<Format Options>] `See below.`

[remdtraj] `Read <filename> as the first replica in a group of replica trajectories.`

  remdtrajtemp <Temperature> | remdtrajidx <#> `Use frames at <Temperature> (for temperature replica trajectories) or index <#> (for Hamiltonian replica trajectories); multiple dimensions are comma-separated.`

  [trajnames <file1>,...,<fileN>] `Do not automatically search for additional replica trajectories; use comma-separated list of trajectory names.`

Read in trajectory specified by filename. See page 581 for currently recognized trajectory file formats. If just the <start> argument is given, all frames from <start> to the last frame of the trajectory will be read. To read in a trajectory with offsets where the last frame # is not known, specify the **last** keyword instead of a <stop> argument, e.g.

```
trajin Test1.crd 10 last 2
```

This will process Test1.crd from frame 10 to the last frame, skipping by 2 frames. To explicitly select only the last frame, specify the **lastframe** keyword:

```
trajin Test1.crd lastframe
```

Here is an example of loading in multiple trajectories which have difference topology files:

```
parm top0.parm7
parm top1.parm7
parm top2.parm7 [top2]
parm top3.parm7
trajin Test0.crd
trajin Test1.crd parm top1.parm7
trajin Test2.crd parm [top2]
trajin Test3.crd parmindex 3
```

Test0.crd is associated with top0.parm7; since no parm was specified it defaulted to the first parm read in. Test1.crd was associated with top1.parm7 by filename, Test2.crd was associated with top2.parm7 by its tag, and finally Test3.crd was associated with top3.parm7 by its index (based on the order it was read in).

Replica Trajectory Processing

If the **remdtraj** keyword is specified the trajectory is treated as belonging to the lowest # replica of a group of REMD trajectories. The remaining replicas can be either automatically detected by following a naming convention of <REMDFILENAME>.X, where X is the replica number, or explicitly specified in a comma-separated list following the **trajnames** keyword. All trajectories will be processed at the same time, but only frames with a temperature matching the one specified by **remdtrajtemp** or **remdtrajidx** will be processed. For example, to process replica trajectories rem.001, rem.002, rem.003, and rem.004, grabbing only the frames at temperature 300.0 (assuming that this is a temperature in the ensemble):

```
trajin rem.001 remdtraj remdtrajtemp 300
```

or

```
trajin rem.001 remdtraj remdtrajtemp 300 trajnames rem.002,rem.003,rem.004
```

Note that the **remdout** keyword is deprecated. For this functionality see the **ensemble** keyword.

28.8.4.1. *Options for Amber NetCDF, Amber NC Restart, Amber Restart:*

```
[usevelascoords] [usefrcascoords]
```

usevelascoords `Read in velocities in place of coordinates if present.`

usefrcascoords `Read in forces in place of coordinates if present.`

28.8.4.2. *Options for CHARMM DCD:*

```
[{ucell | shape}]
```

ucell `Force reading of box information as unit cell (for e.g. NAMD DCD trajectories).`

shape `Force reading of box information as shape matrix.`

28.8.5. trajout

```
trajout <filename> [<format>] [append] [nobox] [novelocity]
        [notemperature] [notime] [noforce] [noreplicadim]
        [parm <parmfile> | parmindex <#>] [onlyframes <range>] [title <title>]
        [onlymembers <memberlist>]
        [start <start>] [stop <stop>] [offset <offset>]
        [ <Format Options> ]
```

<filename> `Trajectory file to write to.`

[<format>] `Keyword specifying output format (see Table on page 581). If not specified format will be determined from extension, otherwise default to Amber trajectory.`

[append] `If <filename> exists, frames will be appended to <filename>.`

[nobox] `Do not write box coordinates to trajectory.`

[novelocity] `Do not write velocities to trajectory.`

[notemperature] `Do not write temperature to trajectory.`

[notime] `Do not write time to trajectory.`

[noreplicadim] `Do not write replica dimensions to trajectory.`

[parm <parmfile>] `Topology filename/tag to associate with trajectory (default first topology).`

[parmindex <#>] `Index of Topology to associate with trajectory (default 0, first topology).`

[onlyframes <range>] `Write only the specified input frames to <filename>.`

[title <title>] `Output trajectory title.`

[onlymembers <memberlist>] `Ensemble processing only; only write from specified members (starting from 0).`

[start <start>] `Begin output at frame <start> (1 by default).`

[stop <stop>] `End output at frame <stop> (last frame by default).`

[offset <offset>] `Skip <offset> frames between each output (1 by default).`

During a run, write frames to trajectory specified by filename in specified file format (Amber trajectory if none specified) after all Action processing has occurred. To write out trajectories within the Action queue see the outtraj Action (28.9.50 on page 629). See page 581 for currently recognized output trajectory formats and their associated keyword(s). Note that now the file type can be determined from the output extension if not specified by a keyword. Multiple output trajectories of any format can be specified.

Frames will be written to the output trajectory when the parameter file being processed matches the parameter file the output trajectory was set up with. So given the input:

```
parm top0.parm7
parm top1.parm7 [top1]
trajin input0.crd
trajin input1.crd parm [top1]
trajout output.crd parm [top1]
```

only frames read in from input1.crd (which is associated with top1.parm7) will be written to output.crd. The trajectory input0.crd is associated with top0.parm7; since no output trajectory is associated with top0.parm7 no frames will be written when processing top0.parm7/input0.crd.

If **onlyframes** is specified, only input frames matching the specified range will be written out. For example, given the input:

```
trajin input.crd 1 10
trajout output.crd onlyframes 2,5-7
```

only frames 2, 5, 6, and 7 from input.crd will be written to output.crd.

28.8.5.1. *Options for pdb format:*

```
[model | multi] [dumpq | parse | vdw] [chainid <ID>]
[pdbres] [pdbatom] [pdbv3] [teradvance]
```

model `(Default) Frames will be written to a single PDB file separated by MODEL/ENDMDL keywords.`

multi `Each frame will be written to a separate file with the frame # appended to <filename>.`

dumpq `PQR format; write charges (in units of e-) and GB radii to occupancy and B-factor columns respectively.`

parse `PQR format; write charges and PARSE radii to occupancy/B-factor columns.`

vdw `PQR format; write charges and vdW radii to occupancy/B-factor columns.`

pdbres: `Use PDB V3 residue names.`

pdbatom: `Use PDB V3 atom names.`

pdbv3: `Use PDB V3 residue/atom names.`

teradvance: `Increment record (atom) number for TER records (not done by default).`

terbyres: `Print TER cards based on residue sequence instead of molecules.`

pdbter: `Print TER cards according to original PDB TER (if available).`

noter: `Do not write TER cards.`

chainid <ID> `Write PDB file with chain ID <ID>.`

sg <group> `Space group for CRYST1 record; only used if box coordinates written.`

include_ep `Include extra points.`

conect `Write CONECT records for all bonds.`

keepext `Keep filename extension; write '<name>.<num>.<ext>' instead (implies 'multi').`

28.8.5.2. *Options for Amber ASCII format:*

```
[remdtraj] [highprecision]
```

remdtraj `Write REMD header to trajectory that includes temperature: 'REMD <Replica> <Step> <Total_Steps> <Temperature>'. Since` *`cpptraj`* `has no concept of replica number, 0 is printed for <Replica>. <Step> and <Total_Steps> are set to the current frame #.`

highprecision: `(EXPERT USE ONLY) Write with 8.6 precision instead of 8.3. Note that since the width does not change, the precision of large coords may be lower than 6.`

28.8.5.3. *Options for Amber NetCDF format:*

```
[remdtraj] [velocity] [force]
```

remdtraj `Write replica temperature to trajectory.`

velocity `Write only velocity information in trajectory.`

force `Write only force information in trajectory.`

mdcrd `Write coordinates to trajectory (only required with mdvel/mdfrc).`

28.8.5.4. *Options for Amber Restart/NetCDF Restart format:*

`[remdtraj] [novelocity] [notime] [time0 <initial time>] [dt <timestep>]`

remdtraj `Write replica temperature to restart. Note that this will automatically include time in the restart file (see the time0 keyword).`

time0 <initial time> `Time for first frame (default 1.0).`

dt <timestep> `Time step between frames (default 1.0). Time is calculated as t=(time0+frame)*dt.`

keepext `Keep filename extension; write '<name>.<num>.<ext>' instead.`

28.8.5.5. *Options for CHARMM DCD:*

`[x64] [ucell]`

x64 `Use 8 byte block size (default 4 bytes).`

ucell `Write older (v21) format trajectory that stores unit cell params instead of shape matrix.`

28.8.5.6. *Options for GROMACS TRX/XTC format:*

`[dt <time step>]`

dt `Time step tp multiply set numbers by (default 1.0). Ignored if time already present.`

28.8.5.7. *Options for mol2 format:*

`[single | multi] [sybyltype] [keepext]`

single `(Default) Frames will be written to a single Mol2 file separated by MOLECULE keywords.`

multi `Each frame will be written to a separate file with the frame # appended to <filename>.`

sybyltype `Convert Amber atom types (if present) to SYBYL types. Requires $AMBERHOME is set.`

sybylatom `File containing Amber to SYBYL atom type correspondance (optional).`

sybylbond `File containing Amber to SYBYL bond type correspondance (optional).`

keepext `Keep filename extension; write '<name>.<num>.<ext>' instead (implies 'multi').`

28.8.5.8. Options for SQM input format:

`[charge <c>]`

charge <c> `Set total integer charge. If not specified it will be calculated from atomic charges.`

28.9. Action Commands

Actions in *cpptraj* operate on frames read in by the ***trajin*** or ***ensemble*** commands one at a time and extract derived data, modify the coordinates/topology in some way, or both. Most Actions in *cpptraj* function exactly the way they do in *ptraj* and are backwards-compatible. Some Action commands in *cpptraj* have extra functionality compared to *ptraj* (such as the per-residue RMSD function of the ***rmsd*** Action, or the ability to write out stripped topologies for visualization in the ***strip*** Action), while other Actions produce slightly different output (like the ***hbond***/***secstruct*** Actions).

Unlike some other command types, when an Action command is issued it is by default added to the Action queue and is not executed until trajectory processing is started (e.g. by a ***run*** or ***go*** command). However, Actions can be executed immediately on COORDS data sets via the ***crdaction*** command (28.5.2 on page 565).

When a frame is modified by an Action, it is modified for every Action that follows them during trajectory processing. For example, given a solvated system with water residues named WAT and the following Action commands:

```
rmsd R1 first :WAT out water-rmsd.dat
strip :WAT
rmsd R2 first :WAT out water-rmsd-2.dat
```

the first ***rms*** command will be valid, but the second ***rms*** command will not since all residues named WAT are removed from the state by the ***strip*** command.

The following Actions are available. If an Action may modify coordinate/topology information for subsequent Actions it is denoted with an X in the `Mod` column.

Command	Description	Mod
angle	Calculate the angle between three points.	
areapermol	Calculate area per molecule for molecules in a specified plane.	
atomiccorr	Calculate average correlation between motions of specified atoms.	
atomicfluct, rmsf	Calculate root mean square fluctuation of specified atoms/residues.	
atommap	Attempt to create a map between atoms in molecules with different atom ordering.	X
autoimage	Automatically re-image coordinates.	X
average	Calculate average structure.	
bounds	Calculate the min/max coordinates for specified atoms. Can be used to create grid data sets.	
box	Set or overwrite box information for frames.	
center	Center specified coordinates to box center or onto reference structure.	X
check, checkoverlap, checkstructure	Check for bad atomic overlaps or bond lengths. Can be used to skip corrupted frames.	
checkchirality	Report chirality around alpha carbons in amino acids (L, D).	
closest, closestwaters	Retain only the specified number of solvent molecules closest to specified solute.	X
clusterdihedral	Assign frames into clusters based on binning of backbone dihedral angles in amino acids.	
contacts	Older version of ***nativecontacts***, retained for backwards compatibility.	
createcrd	Create a COORDS data set from input frames.	
createreservoir	Create a structure reservoir for use with reservoir REMD simulations.	
density	Calculate density along a coordinate.	
diffusion	Calculate translational diffusion of molecules.	
dihedral	Calculate the dihedral angle using four points.	
dipole	Bin dipoles of solvent molecules in 3D grid. Not well tested, may be obsolete.	
distance	Calculate the distance between two points.	
drms, drmsd	Calculate the RMSD of distance pairs within selected atoms.	
dssp, secstruct	Calculate secondary structure content using the DSSP algorithm	
energy	Calculate simple bond, angle, dihedral, and non-bonded energy terms (no PME).	
esander	Calculate energies using via SANDER; requires compilation with the SANDER API.	
filter	Filter frames for subsequent Actions using data sets and user defined criteria.	
fixatomorder	Fix atom ordering so that all atoms in molecules are sequential.	X

gist	Perform grid inhomogenous solvation theory.	
grid	Bin selected atoms on a 3D grid.	
hbond	Calculate hydrogen bonds using geometric criteria.	
image	Re-image coordinates. The ***autoimage*** command typically provides better results.	X
jcoupling	Calculate J-coupling values from specified dihedral angles.	
lessplit	Split/average frames from LES trajectories.	
lie	Calculate linear interaction energy between user-specified ligand and surroundings.	
lipidorder	Calculate order parameters for lipids in planar membranes.	
makestructure	Modify structure by applying dihedral values to specified residues.	X
mask	Print the results of selection by specified atom mask. Good for distnace-based masks.	
matrix	Calculate a matrix of the specified type from input coordinates.	
minimage	Calculate minimum non-self imaged distance between atoms in specified masks.	
molsurf	Calculate Connolly surface area of specified atoms. Cannot do partial surface areas.	
multidihedral	Calculate multiple dihedral angles of specified/given types.	
multivector	Calculate multiple vectors between specified atoms.	
nastruct	Perform nucelic acid structure analysis.	
nativecontacts	Calculate native contacts within a region or between two regions using a given reference. Can also be used to get min/max distances between groups of atoms.	
outtraj	Write frames to a trajectory file within a list of Actions.	
pairdist	Calculate pair distribution function.	
pairwise	Calculate pair-wise non-bonded energies.	
principal	Calculate and optionally align system along principal axes.	X
projection	Project coordinates along given eigenvectors.	
pucker	Calculate ring pucker using five or six points.	
radgyr, rog	Calculate radius of gyration (and optionally tensor) for specified atoms.	
radial, rdf	Calculate radial distribution function.	
randomizeions	Swap specified ions with randomly selected solvent molecules.	X
replicatecell	Replicate unit cell in specified (or all) directions for specfied atoms and write to trajectory.	
rms, rmsd	Perform best fit of coordinates to reference and calculate coordinate RMSD. Fitting can be disabled.	X
rotate	Rotate the system around X/Y/Z axes, a specified axis, or via given rotation matrices.	X
runavg, runningaverage	Calculate the running average of coordinates over specified window size.	X
scale	Scale coordinates in X/Y/Z directions by specified factors.	X
setvelocity	Set velocities for specified atoms using Maxwellian distribution based on given temperature.	
spam	SPAM method for estimating relative free energies of waters in hydration shell around proteins.	X
stfcdiffusion	Alternative translational diffusion calculation which can calculate diffusion in specified regions.	
strip	Remove specified atoms from the system.	
surf	Calculate the LCPO surface area of specified atoms. Can do partial surface areas.	
symmrmsd	Calculate symmetry-corrected RMSD.	X

temperature	Calculate system temperature using velocities of specified atoms.	
trans, translate	Translate specified atoms by specified amounts in X/Y/Z directions.	X
unstrip	Undo all previous ***strip*** Action commands.	
unwrap	Reverse of ***image***; unwrap selected atoms so they have continuous trajectories.	X
vector	Calculate various types of vector quantities.	
velocityautocorr	Calculate velocity autocorrelation function.	
volmap	Create volumetric map for specified coordinates; similar to ***grid*** but takes into account atomic radii. Similar to VMD ***volmap***.	
volume	Calculate unit cell volume.	
watershell	Calculate the number of waters in the first and second solvation shells based on distance critera.	

28.9.1. angle

```
angle [<dataset name>] <mask1> <mask2> <mask3> [out <filename>] [mass]
```

[<dataset name>] `Output data set name.`

<maskX> `Three atom masks selecting atom(s) to calculate angle for.`

[out <filename>] `Output file name.`

[mass] `Use center of mass of atoms in <maskX> instead of geometric center.`

Calculate angle (in degrees) between atoms in <mask1>, <mask2>, and <mask3>. For example, to calculate the angle between the first three atoms in the system:

```
angle A123 @1 @2 @3 out A123.agr
```

28.9.2. areapermol

```
areapermol [<name>] {[<mask1>] [nlayers <#>] | nmols <#>} [out <filename>] [{xy | xz | yz}]
```

[<name>] `Data set name.`

[<mask1>] `Atom mask for selecting molecules.  If any atom in a molecule is selected the whole molecule is selected.`

[nlayers <#>] `Number of layers of molecules.  Total number of molecules used will be # molecules divided by # layers.`

[nmols <#>] `If <mask1> is not specified, the number of molecules to use when calculating area per molecule.`

[out <filename>] `Output file name.`

[{xy|xz|yz}] `Cross-section of box to calculate area of.  Default is X-Y.`

Calculate area per molecule as Area / # molecules. The area is determined from the specified cross-section of the box (X-Y by default). Currently the calculation is only guaranteed to work properly with orthorhombic unit cells. For example, to get the area per molecule of residues named “OL” which are arranged in 2 layers:

```
areapermol OL_area :OL nlayers 2 out apm.dat
```

28.9.3. atomiccorr

```
atomiccorr [<mask>] out <filename> [cut <cutoff>] [min <min spacing>]
           [byatom | byres]
```

<mask> `Atoms to calculate motion vectors for.`

out <filename> `File to write results to.`

cut <cutoff> `Only print correlations with absolute value greater than <cutoff>.`

min <min spacing> `Only calculate correlations for motion vectors spaced <min spacing> apart.`

byatom `Default; calculate atomic motion vectors.`

byres `Calculate motion vectors for entire residues (selected atoms in residues only).`

Calculate average correlations between the motion of atoms in <mask>. For each frame, a motion vector is calculated for each selected atom from its previous position to its current position. For each pair of motion vectors *Va* and *Vb*, the average correlation between those vectors is calculated as the average of the dot product of those vectors over all *N* frames.

$$AvgCorr(a,b) = \frac{\sum V_a(i) \cdot V_b(i)}{N}$$

The value of AvgCorr can range from 1.0 (correlated) to 0.0 (no correlation) to -1.0 (anti-correlated). For example, to calculate the correlation of motion vectors between residues 1 to 13, writing to a Gnuplot-readable formatted file:

```
atomiccorr :1-13 out acorr.gnu byres
```

28.9.4. atomicfluct | rmsf

```
atomicfluct [out <filename>] [<mask>] [byres | byatom | bymask] [bfactor]
            [calcadp [adpout <file>]]
            [start <start>] [stop <stop>] [offset <offset>]
```

out <filename> `Write data to file named <filename>`

[<mask>] `Calculate fluctuations for atoms in <mask> (all if not specified).`

byres `Output the average (mass-weighted) fluctuation by residue.`

bymask `Output the average (mass-weighted) fluctuation for all atoms in <mask>.`

byatom (default) `Output the fluctuation by atom.`

[bfactor] Calculate atomic positional fluctuations squared and weight by $\frac{8}{3}\pi^2$; this is similar but not necessarily equivalent to the calculation of crystallographic B-factors.

[calcadp [adpout <file>]] `Calculate anisotropic displacement parameters and optionally output them to <file>.`

[<start>] `Frame to begin calculation at (default 1).`

[<stop>] `Frame to end calculation at (default last).`

[<offset>] `Frames to skip between calculations (default 1).`

Compute the atomic positional fluctuations (also referred to as root-mean-square fluctuations, RMSF) for atoms specified in the **<mask>**. Note that RMS fitting is not done implicitly. If you want fluctuations without rotations or translations (for example to the average structure), perform an RMS fit to the average structure (best) or the first structure (see ***rmsd***) prior to this calculation. The units are (Å) for RMSF or $Å^2 \times \frac{8}{3}\pi^2$ if **bfactor** is specified.

If **byres** or **bymask** are specified, the mass-weighted average of atomic fluctuations of each atom for either each residue or the entire mask will be calculated respectively:

$$\langle Fluct \rangle = \frac{\sum AtomFluct_i * Mass_i}{\sum Mass_i}$$

If **calcadp** is specified, anisotropic displacement factors for atoms will be calculated and written to the file specified by **adpout** (or STDOUT if not specified) using PDB ANISOU record format. Note that **calcadp** automatically implies **bfactor**.

With *cpptraj* it is possible to perform coordinate averaging, the fit to average coordinates, and the atomic fluctuation calculation in a single execution like so:

```
parm myparm.parm7
trajin mytrajectory.crd
rms first
average crdset MyAvg
run
rms ref MyAvg
atomicfluct out fluct.agr
```

To write the mass-weighted B-factors for the protein backbone atoms C, CA, and N, averaged by residue use the command:

```
atomicfluct out back.agr @C,CA,N byres bfactor
```

To write the RMSF or atomic positional fluctuations of the same atoms, use the command:

```
atomicfluct out backbone-atoms.agr @C,CA,N
```

28.9.5. atommap

```
atommap <target> <reference> [mapout <filename>] [maponly]
        [rmsfit [ rmsout <rmsout> ]]
```

<target> `Reference structure whose atoms will be remapped.`

<reference> `Reference structure that <target> should be mapped to.`

mapout <filename> `Write atom map to <filename> with format:`
`TargetAtomNumber TargetAtomName ReferenceAtomNumber ReferenceAtomName`
`Target atoms that cannot be mapped to a reference atom are denoted "--".`

maponly `Write atom map but do not reorder atoms.`

rmsfit `Any input frames using the same topology as <target> will be RMS fit to <reference> using whatever atoms could be mapped.`

rmsout <rmsout> `If rmsfit specified, write resulting RMSDs to <rmsout>.`

Attempt to map the atoms of <target> to those of <reference> based on structural similarity. This is useful e.g. when there are two files containing the same structure but with different atom names or atom ordering. Both <target> and <reference> need to have been read in with a previous ***reference*** command. The state will then be modified so that any trajectory read in with the same parameter file as <target> will have its atoms mapped (i.e. reordered) to match those of <reference>. If the number of atoms that can be mapped in <target> are less than those in <reference>, the reference structure specified by <reference> will be modified to include only mapped atoms; this is useful if for example the reference structure is protonated with respect to the target. The **rmsfit** keyword is useful in cases where the atom mapping will not be complete (e.g. two ligands with the same scaffold but different substituents).

For example, say you have the same ligand structure in two files, Ref.mol2 and Lig.mol2, but the atom ordering in each file is different. To map the atoms in Lig.mol2 onto those of Ref.mol2 so that Lig.mol2 has the same ordering as Ref.mol2:

```
parm Lig.mol2
reference Lig.mol2
parm Ref.mol2
reference Ref.mol2 parmindex 1
atommap Lig.mol2 Ref.mol2 mapout atommap.dat
trajin Lig.mol2
trajout Lig.reordered.mol2 mol2
```

28.9.6. autoimage

```
autoimage [<mask> | anchor <mask> [fixed <mask>] [mobile <mask>]]
          [origin] [firstatom] [familiar | triclinic]
```

[<mask> | anchor <mask>] `Atoms to image around; this is the region that will be centered. Default is the entire first molecule.`

[fixed <mask>] `Molecules that should remain 'fixed' to the anchor region; default is all non-ion/non-solvent molecules.`

[mobile <mask>] `Molecules that can be freely imaged; default is all ion/solvent molecules.`

[origin] `Center anchor region at the origin; if not specified, center at box center.`

[firstatom] `Image based on molecule first atom; default is to image by molecule center of mass.`

[familiar] `Image to familiar truncated-octahedral shape; this is on by default if the original cell is truncated octahedron.`

[triclinic] `Force general triclinic imaging.`

Automatically center and image (by molecule) a trajectory with periodic boundaries. For most cases just specifying ***'autoimage'*** alone is sufficient. The atoms of the **'anchor'** region (default the entire first molecule) will be centered; all **'fixed'** molecules will be imaged only if imaging brings them closer to the **'anchor'** molecule (default for **'fixed'** molecules is all non-solvent non-ion molecules). All other molecules (referred to as **'mobile'**) will be imaged freely.

The autoimage command works for the majority of systems; however, for very densely packed systems the default anchor (entire first molecule) may not be appropriate. In these cases, it is recommended to choose as the anchor a small region which should lie near the center of your system. For example, in a protein dimer system one could choose a single residue that is near the center of the interface between the two monomers.

28.9.7. average

```
average {crdset <set name> | <filename>} [<mask>]
        [start <start>] [stop <stop>] [offset <offset>]
        [Trajout Args]
```

<filename> `If specified, write averaged coordinates to <filename> (not compatible with crdset).`

crdset <set name> `If specified, save averaged coordinates to COORDS set <set name> (not compatible with <filename>).`

[<mask>] `Average coordinates in <mask> (all atoms if not specified).`

[<start>] `Frame to begin calculation at (default 1).`

[<stop>] `Frame to end calculation at (default last).`

[<offset>] `Frames to skip between calculations (default 1).`

[Trajout args] `Output trajectory format argument(s) (default Amber Trajectory).`

Calculate the average of input coordinates and write out to file named **<filename>** or save to COORDS set named **<set name>** in any trajectory format *cpptraj* recognizes (Amber Trajectory if not specified). If the number of atoms in **<mask>** are less than the total number of atoms, the topology will be stripped to match **<mask>**.

Note that since coordinates are being averaged over many frames, resulting structures may appear distorted. For example, if one averages the coordinates of a freely rotating methyl group the average position of the hydrogen atoms will be close to the center of rotation. Also note that typically one will want to remove global rotational and translation movement prior to this command by using e.g. the ***rms*** (28.9.60 on page 635) command.

Any arguments that are valid for the ***trajout*** command (28.8.5 on page 584) can be passed to this command in order to control the format of the output coordinates. For example, to write out a PDB file containing the averaged coordinates over all frames:

```
average test.pdb pdb
```

To write out a mol2 file containing only the averaged coordinates of residues 1 to 10 for frames 1 to 100:

```
average test.mol2 mol2 start 1 stop 100 :1-10
```

To create an average structure of atoms named CA and then use it as a reference for an rms command in a subsequent run:

```
trajin Input.nc
average crdset MyAvg @CA
run
rms ref MyAvg @CA out RmsToAvg.dat
run
```

28.9.8. avgcoord

This command is deprecated. Use 'vector center' (optionally with keyword 'magnitude') instead.

28.9.9. bounds

```
bounds [<mask>] [out <filename>]
       [dx <dx> [dy <dy>] [dz <dz>] name <gridname> [offset <bin offset>]]
```

[<mask>] `Mask of atoms to determine bounds of.`

[out <filename>] `File to write bounds to (default STDOUT if not specified).`

[dx <dx> [dy <dy>] [dz <dz>]] `Triggers creation of a grid data set from bounds. Spacings of generated grid in the X, Y and Z directions. If only dx is specified <dx> will be used for <dy> and <dz> as well.`

[name <gridname>] `Name of generated grid data set.`

[offset <bin offset>] `Number of bins to add/subtract in each direction to generated grid.`

Calculate the boundaries (i.e. the max/min X/Y/Z coordinates) of atoms in **<mask>** and write to **<filename>** (STDOUT if not specified). Useful for determining dimensions for the ***grid*** command, and can be used to generate a grid data set that can be used by ***grid*** (see 28.9.33 on page 611).

28.9.10. box

```
box [x <xval>] [y <yval>] [z <zval>] [alpha <a>] [beta <b>] [gamma <g>]
    [nobox] [truncoct]
```

[x <xval>] [y <yval>] [z <zval>] Change box length(s) to specified value(s).

[alpha <a>] [beta <b>] [gamma <g>] Change box angle(s) to specified value(s).

[nobox] Remove any existing box information.

[truncoct] Set box angles to truncated octahedron.

Modify box information during trajectory processing. Note that this will permanently modify the box information for topology files during trajectory processing as well. It is possible to modify any number of the box parameters (e.g. only the Z length can be modified if desired while leaving all other parameters intact).

28.9.11. center

```
center [<mask>] [origin] [mass]
       [ reference | ref <name> | refindex <#> [<refmask>]]
```

[<mask>] Center based on atoms in mask; default is all atoms.

[origin] Center to origin (0, 0, 0); default is center to box center (X/2, Y/2, Z/2).

[mass] Use center of mass instead of geometric center.

[reference | ref <name> | refindex <#> [<refmask]] Center using coordinates in specified reference structure selected by <refmask> (<mask> if not specified.

Move all atoms so that the center of the atoms in **<mask>** is centered at the specified location: box center (default), coordinate origin, or reference coordinates.

For example, to move all coordinates so that the center of mass of residue 1 is at the center of the box:

```
center :1 mass
```

28.9.12. check | checkoverlap | checkstructure

```
check [<mask>] [around <mask2>] [reportfile <report>] [noimage] [skipbadframes]
      [offset <offset>] [cut <cut>] [nobondcheck] [silent]
```

[<mask>] Check structure of atoms in <mask> (all if not specified).

[around <mask2>] If specified, only check for problems between atoms in <mask> and atoms in <mask2>.

[reportfile <report>] Write any problems found to <report> (STDOUT if not specified).

[noimage] Do not image distances.

[skipbadframes] If errors are encountered for a frame, subsequent actions/trajectory output will be skipped.

[offset <offset>] Report bond lengths greater than the equilibrium value plus <offset> (default 1.0 Å)

[cut <cut>] Report atoms closer than <cut> (default 0.8 Å).

[nobondcheck] Check overlaps only.

[silent] Do not print information for bad frames - useful in conjunction with the skipbadframes option.

Check the structure and report problems related to atomic overlap/unusual bond length. Problems are reported when any two atoms in the mask are closer than **<cut>**. If bonds are being checked then bond lengths greater than their equilibrium value + **<offset>** are reported as well. This command can also be used to skip corrupted frames in a trajectory during processing. For example, if this message is encountered:

```
Warning: Frame 10 coords 1 & 2 overlap at origin; may be corrupt.
```

One could use ***check*** so that e.g. a subsequent ***distance*** command is not processed for bad frames:

```
check @1,2 skipbadframes silent
distance d1 :1 :10
```

Usually frame corruption can be detected using only a few atoms, but this may not catch all types of corruption. The more atoms that are used the better the corruption detection will be, but the slower it will be to process the command. Typically a good procedure to follow when corruption is suspected is to run ***check*** using all important atoms (e.g. all solute heavy atoms) with the **skipbadframes** keyword followed by a ***trajout*** command to write all non-corrupt frames, for example:

```
trajin corrupted.crd
check :1-13 skipbadframes silent
trajout fixed.corrupted.nc
```

28.9.13. checkchirality

```
checkchirality [<name>] [<mask>] [out <filename>]
```

[<name>] `Data set name.`

[<mask>] `Atoms to check.`

[out <filename>] `File to write results to.`

`DataSet Aspects:`

[L] `Number of frames 'L' for each residue.`

[D] `Number of frames 'D' for each resdiue.`

Check the chirality around the alpha carbon in amino acid residues selected by <mask>. Note that cpptraj expects atom names to correspond to the PDB V3 standard: N, CA, C, CB. For each residue, the number of frames in which the amino acid is 'L' or 'D' will be recorded. For example, to check the chirality of all amino acids in a system and write to a file named chiral.dat with data set name DPDP:

```
checkchirality DPDP out chiral.dat
```

Output will have format similar to:

```
#Res      DPDP[L]  DPDP[D]
   2.000      100        0
```

So in this example residue 2 was 'L' for 100 frames and 'D' for 0 frames.

28.9.14. closest | closestwaters

```
closest <# to keep> <mask> [noimage] [first | oxygen] [center]
        [closestout <filename>] [name <setname>] [outprefix <parmprefix>]
```

<# to keep> `Number of solvent molecules to keep around <mask>`

<mask> `Mask of atoms to search for closest waters around.`

[noimage] `Do not perform imaging; only recommended if trajectory has previously been imaged.`

[first | oxygen] `Calculate distances between all atoms in <mask> and the first atom of solvent only (recommended for standard water models as it will increase speed of calculation).`

[center] `Search for waters closest to center of <mask> instead of each atom in <mask>.`

[closestout <filename>] `Write information on the closest solvent molecules to <filename>.`

[outprefix <prefix>] `Write corresponding topology to file with name prefix <prefix>.`

`DataSet Aspects:`

[Frame] `Frame number.`

[Mol] `Original solvent molecule number.`

[Dist] `Solvent molecule distance in Å.`

[FirstAtm] `First atom number of original solvent molecule.`

Similar to the ***strip*** command, but modify coordinate frame and topology by keeping only the specified number of closest solvent molecules to the region specified by the given mask. Solvent molecules can be determined automatically by *cpptraj* (by default residues named WAT, HOH, or TIP3) or can be specified prior via the ***solvent*** command (28.7.16 on page 580). The format of the **closestout** file is:

```
Frame    Molecule    Distance    FirstAtom#
```

For example, to obtain the 10 closest waters to residues 1-268 by distance to the first atom of the waters, write out which waters were closest for each frame to a file called "closestmols.dat", and write out the stripped topology with prefix "closest" containing only the solute and 10 waters:

```
closest 10 :1-268 first closestout closestmols.dat outprefix closest
```

28.9.15. cluster

Although the **'cluster'** command can still be specified as an action, it is now considered an analysis. See 28.10.4 on page 652.

28.9.16. clusterdihedral

```
clusterdihedral [phibins <N>] [psibins <M>] [out <outfile>]
                [dihedralfile <dfile> | <mask>]
                [framefile <framefile>] [clusterinfo <infofile>]
                [clustervtime <cvtfile>] [cut <CUT>]
```

Cluster frames in a trajectory using dihedral angles. To define which dihedral angles will be used for clustering either an atom mask or an input file specified by the **dihedralfile** keyword should be used. If dihedral file is used, each line in the file should contain a dihedral to be binned with format:

```
ATOM#1 ATOM#2 ATOM#3 ATOM#4 #BINS
```

where the ATOM arguments are the atom numbers (starting from 1) defining the dihedral and #BINS is the number of bins to be used (so if #BINS=10 the width of each bin will be 36º). If an atom mask is specified, only protein backbone dihedrals (Phi and Psi defined using atom names C-N-CA-C and N-CA-C-N) within the mask will be used, with the bin sizes specified by the phibins and psibins keywords (default for each is 10 bins).

Output will either be written to STDOUT or the file specified by the **out** keyword. First, information about which dihedrals were clustered will be printed. Then the number of clusters will be printed, followed by detailed information of each cluster. The clusters are sorted from most populated to least populated. Each cluster line has format

```
Cluster CLUSTERNUM CLUSTERPOP [ dihedral1bin, dihedral2bin ... dihedralNbin ]
```

followed by a list of frame numbers that belong to that cluster. If a cutoff is specified by **cut**, only clusters with population greater than CUT will be printed.

If specified by the **clustervtime** keyword, the number of clusters for each frame will be printed to <cvtfile>. If specified by the **framefile** keyword, a file containing cluster information for each frame will be written with format

```
Frame CLUSTERNUM CLUSTERSIZE DIHEDRALBINID
```

where DIHEDRALBINID is a number that identifies the unique combination of dihedral bins this cluster belongs to (specifically it is a 3*number-of-dihedral-characters long number composed of the individual dihedral bins).

If specified by the **clusterinfo** keyword, a file containing information on each dihedral and each cluster will be printed. This file can be read by SANDER for use with REMD with a structure reservoir (-rremd=3). The file, which is essentially a simplified version of the main output file, has the following format:

```
#DIHEDRALS
dihedral1_atom1 dihedral1_atom2 dihedral1_atom3 dihedral1_atom4
...
#CLUSTERS
CLUSTERNUM1 CLUSTERSIZE1 DIHEDRALBINID1
...
```

28.9.17. contacts

```
contacts [ first | reference | ref <ref> | refindex <#> ] [byresidue]
         [out <filename>] [time <interval>] [distance <cutoff>] [<mask>]
```

NOTE: Users are encouraged to try the ***nativecontacts*** command (on page 627), an update version of this command.

For each atom given in *mask*, calculate the number of other atoms (contacts) within the distance *cutoff*. The default cutoff is 7.0 A. Only atoms in *mask* are potential interaction partners (e.g., a mask @CA will evaluate only contacts between CA atoms). The results are dumped to *filename* if the keyword “`out`” is specified. Thereby, the time between snapshots is taken to be *interval*. In addition to the number of overall contacts, the number of native contacts is also determined. Native contacts are those that have been found either in the first snapshot of the trajectory (if the keyword “`first`” is specified) or in a reference structure (if the keyword “`reference`” is specified). Finally, if the keyword “`byresidue`” is provided, results are output on a per-residue basis for each snapshot, whereby the number of native contacts is written to *filename*.native.

28.9.18. createcrd

```
createcrd [<name>] [ parm <name> | parmindex <#> ]
```

This command creates a COORDS data set named <name> using trajectory frames that are associated with the specified topology.

For example, to save frames that have been previously RMS-fit to a reference structure into a COORDS set named MyCrd you would use the input:

```
rms reference :1-12@CA
createcrd MyCrd
strip :6-8
```

Note that here the ***strip*** command will have no effect on the coordinates saved in MyCrd since it occurs after the ***createcrd*** command.

28.9.19. createreservoir

```
createreservoir <filename> ene <energy data set> [bin <cluster bin data set>]
                temp0 <temp0> iseed <iseed> [velocity]
                [parm <parmfile> | parmindex <#>] [title <title>]
```

<filename> File name of the reservoir to create.

ene <energy data set> Data set with energies corresponding to frames.

[bin <cluster bin data set>] Data set with bin numbers (for RREMD=3).

temp0 <temp0> Reservoir temperature.

iseed <iseed> Reservoir random number seed.

[velocity] Include velocities in the reservoir.

[parm <parmfile> | parmindex <#>] Associated topology.

[title <title>] Reservoir title.

Create structure reservoir for use with reservoir REMD simulations using energies in <energy data set>, temperature <temp0> and random seed <iseed> Include velocities if [velocity] is specified. If <cluster bin data set> is specified from e.g. a previous 'clusterdihedral' command, the reservoir can be used for non-Boltzmann reservoir REMD (rremd==3).

28.9.20. density

```
density [out <filename>] [name <set name>] [delta <resolution>] [x|y|z]
        [number|mass|charge|electron] [efile <filename>]
        <mask1> ... <maskN>
```

out Output file for histogram: relative distances vs. densities for each mask.

delta Resolution, i.e. determines number of slices. (0.25 Å)

x|y|z Coordinate for density calculation. (z)

number|mass|charge|electron Number, mass, partial charge (q) or electron (Ne - q) density. To convert the electron density to e-/Å3 divide by the (average) area spanned by the other two dimensions. (number)

mask1 ... maskN Arbitrary number of masks for atom selection; a dataset is created and the output will contain entries for each mask.

DataSet Aspects:

[avg] Average density over coordinate.

[sd] Standard deviation of density over coordinate.

Calculate specified density for system. Defaults are shown in parentheses above. The format of the file is as follows. Comments are lines starting with '#' or empty lines. All other lines must contain the atom type followed by an integer number for the electron number. Entries must be separated by spaces or '='. Example input:

```
density out number_density.dat number delta 0.25 ":POPC@P1" ":POPC@N" \
        ":POPC@C2" ":POPC"
density out mass_density.dat mass delta 0.25 ":POPC@P1" ":POPC@N" \
        ":POPC@C2" ":POPC"
density out charge_density.dat charge delta 0.25 ":POPC@P1" ":POPC@N" \
        ":POPC@C2" ":POPC"
density out electron_density.dat electron delta 0.25 efile Nelec.in \
        ":POPC@P1" ":POPC@N" ":POPC@C2" ":POPC" ":TIP3" \
        ":POPC | :TIP3" "*"
density out ion_density.dat number delta 0.25 ":SOD" ":CLA"
```

See also $AMBERHOME/AmberTools/test/cpptraj/Test_Density.

28.9.21. diffusion

Note that although the syntax for ***diffusion*** *has changed as of version 16, the old syntax is still supported.*

```
diffusion [{out <filename> | separateout <suffix>}] [time <time per frame>] [noimage]
       [<mask>] [<set name>] [individual] [diffout <filename>] [nocalc]
```

[out <filename>] `Write mean-square displacement (MSD) data set output to file specified by <filename>.`

[separateout <suffix>] `Write each MSD data set type to files with suffix <suffix>; see description below.`

[time <time_per_frame>] `Time in-between each coordinate frame in ps; default is 1.0.`

[noimage] `If specified do not perform imaging.  If this is specified coordinates should be unwrapped prior to this command.`

[<mask>] `Mask of atoms to calculate diffusion for; default all atoms.`

[<set name>] `MSD data set name.`

[individual] `Write diffusion for each individual atom as well as average diffusion for atoms in mask.`

[diffout <filename>] `Write diffusion contants calculated from fits of MSD data sets to <filename>.`

[nocalc] `Do not calculate diffusion constants.`

`DataSet Aspects:`

[X] `MSD(s) in the X direction.`

[Y] `MSD(s) in the Y direction.`

[Z] `MSD(s) in the Z direction.`

[R] `Overall MSD(s).`

[A] `Overall displacement(s).`

[D] `Diffusion constants.`

[Label] `Diffusion constant lablels.`

[Slope] `Linear regression slopes.`

[Intercept] `Linear refression Y-intercepts.`

[Corr] `Linear regression correlation coefficients.`

Compute mean square displacement (MSD) plots (using distance traveled from initial position) for the atoms in **<mask>**. By default only the diffusion averaged over all atoms in **<mask>** is calculated; if **individual** is specified diffusion for individual atoms is calculated as well.

In order to correctly calculate diffusion molecules should take continuous paths, so imaging of atoms is autoimatically performed. If the trajectory is already unwrapped (or the unwrap command is used prior to this command) the **noimage** keyword can be used.

The following types of displacements are calculated. If **separateout** is specfied the following files will be created:

x_<suffix> Mean square displacement(s) in the X direction (in Å^2/ps).

y_<suffix> Mean square displacement(s) in the Y direction (in Å^2/ps).

z_<suffix> Mean square displacement(s) in the Z direction (in Å^2/ps).

r_<suffix> Overall mean square displacement(s) (in Å^2/ps).

a_<suffix> Total distance traveled (in Å/ps).

The diffusion coefficient ***D*** can be calculated using the Einstein relation:

$$2nD = \lim_{t\to\infty} \frac{MSD}{t}$$

Where ***n*** is the number of dimensions; for overall MSD n = 3, for single dimension MSD (e.g. X) n = 1, etc. Unless **nocalc** is specified, the diffusion constant is calculated automatically from MSD data sets (and written to the file specified by **diffout**) in the following manner. The slope the plot of MSD versus time is obtained via linear regression. To convert from units of Å^2/ps to 1x10^{-5} cm^2/s, the slope is multiplied by 10.0/(2n). Both the calculated diffusion constants as well as the results of the fit are reported.

Due to the fact that diffusion is currently calculated from initial positions only, diffusion calculated for small numbers of atoms will be inherently stochastic, so the results are most sensible when averaged over many atoms; for example, the diffusion of water should be calculated using all waters in the system.

For example, to calculate the diffusion of water in a system:

```
diffusion :WAT@O out WAT_O.agr WAT_O diffout DC.dat
```

28.9.22. dihedral

```
dihedral [<name>] <mask1> <mask2> <mask3> <mask4> [out <filename>] [mass]
         [type {alpha|beta|gamma|delta|epsilon|zeta|chi|c2p|h1p|phi|psi|omega|pchi}]
         [range360]
```

[<name>] `Output data set name.`

<maskX> `Four atom masks selecting atom(s) to calculate dihedral for.`

[out <filename>] `Output file name.`

[mass] `Use center of mass of atoms in <maskX>; default is geometric center.`

[range360] `Output dihedral angle values from 0 to 360 degrees instead of -180 to 180 degrees.`

[type <type>] `Label dihedral as <type> for use with` ***statistics*** `analysis; note 'chi' is nucleic acid chi and 'pchi' is protein chi.`

Calculate dihedral angle (in degrees) between the planes defined by atoms in <mask1>, <mask2>, <mask3> and <mask2>, <mask3>, <mask4>.

28.9.23. dihedralscan

This command has been replaced by ***permutedihedrals***; see 28.5.7 on page 566. To calculate multiple dihedral angles see the ***multidihedral*** command.

28.9.24. dipole

```
dipole <filename> {data <dsname> | <nx> <dx> <ny> <dy> <nz> <dz> [gridcenter <cx> <cy> <cz>]}
    [box|origin|center <mask>] [negative] [name <gridname>]
       <mask1> {origin | box} [max <max_percent>]
```

NOTE: This command is not well-tested and may be obsolete.

Same as **grid** (see 28.9.33 on page 611 below) except that dipoles of the solvent molecules are binned. The output file format is for Chris Bayly's discern delegate program that comes with Midas/Plus. Consult the code in Action_Dipole.cpp for more information.

28.9.25. distance

```
distance [<name>] <mask1> <mask2> [out <filename>] [geom] [noimage] [type noe]
  Options for 'type noe':
    [bound <lower> bound <upper>] [rexp <expected>] [noe_strong] [noe_medium] [noe_weak]
```

[<name>] `Output data set name`

<maskX> `Two atom masks selecting atom(s) to calculate distance between.`

[out <filename>] `Output filename.`

[geom] `Use geometric center of atoms in <mask1>/<mask2>; default is to use center of mass.`

[noimage] `Do not image distances across periodic boundaries.`

[type noe] `Mark distance as 'noe' for use with` ***`statistics`*** `analysis.`

  [bound <lower> bound <upper>] `Lower and upper bounds for NOE (in Angstroms); must specify both.`

  [rexp <expected>] `Expected value for NOE (in Angstroms); if not given '(<lower> + <upper>)' / 2.0 is used.`

  [noe_strong] `Set lower and upper bounds to 1.8 and 2.9 Å respectively.`

  [noe_medium] `Set lower and upper bounds to 2.9 and 3.5 Å respectively.`

  [noe_weak] `Set lower and upper bounds to 3.5 and 5.0 Å respectively.`

Calculate distance between the center of mass of atoms in <mask1> to atoms in <mask2>. If **geom** is specified use the geometric center instead. For periodic systems imaging is turned on by default; the **noimage** keyword disables imaging.

A distance can be labeled using '`type noe`' for further analysis as an NOE using the ***'statistics'*** analysis command.

28.9.26. drms | drmsd (distance RMSD)

```
drmsd [<dataset name>] [<mask> [<refmask>]] [out <filename>]
      [ first | ref <refname> | refindex <#> |
        reftraj <trajname> [parm <trajparm> | parmindex <parm#>] ]
```

[<dataset name>] `Output data set name.`

[<mask>] `Atoms to calculate DRMSD for.`

[<refmask>] `Mask corresponding to atoms in reference; if not specified, <mask> is used.`

[out <filename>] `Output file name.`

[first] `Use the first trajectory frame processed as reference.`

[reference] `Use the first previously read in reference structure.`

[ref <refname>] `Use previously read in reference structure specified by <refname>.`

[refindex <#>] `Use previously read in reference structure specified by <#> (based on order read in).`

previous `Use frame prior to current frame as reference.`

reftraj <name> `Use frames from COORDS set <name> or read in from trajectory file <name> as references. Each frame from <name> is used in turn, so that frame 1 is compared to frame 1 from <name>, frame 2 is compared to frame 2 from <name> and so on. If <trajname> runs out of frames before processing is complete, the last frame of <trajname> continues to be used as the reference.`

parm <parmname> | parmindex <#> `If reftraj specifies a file associate trajectory <name> with specified topology; if not specified the first topology is used.`

Calculate the distance RMSD (i.e. the RMSD of all pairs of internal distances) between atoms in the frame defined by **<mask>** (all if no **<mask>** specified) to atoms in a reference defined by **<refmask>** (**<mask>** if **<refmask>** not specified). Both **<mask>** and **<refmask>** must specify the same number of atoms, otherwise an error will occur.

Because this method compares pairs of internal distances and not absolute coordinates, it is not sensitive to translations and rotations the way that a no-fit RMSD calculation is. It can be more time consuming however, as (N^2-N)/2 distances must be calculated and compared for both the target and reference structures.

For example, to get the DRMSD of a residue named LIG to its structure in the first frame read in:

```
drmsd :LIG first out drmsd.dat
```

28.9.27. dssp

See 28.9.68 on page 638.

28.9.28. energy

```
energy [<name>] [<mask1>] [out <filename>]
    [bond] [angle] [dihedral] [nb14] {[nonbond] | [elec] [vdw]}
       [ etype {simple |
                directsum [npoints <N>] |
                ewald [cut <cutoff>] [dsumtol <dtol>] [rsumtol <rtol>]
                [ewcoeff <coeff>] [skinnb <skinnb>]
                [mlimits <X>,<Y>,<Z>]} ]
```

[<name>] `Data set name.`

[<mask1>] `Mask of atoms to calculate energy for.`

[out <filename>] `File to write results to.`

[bond] `Calculate bond energy.`

[angle] `Calculate angle energy.`

[dihedral] `Calculate dihedral energy.`

[nb14] `Calculate nonbonded 1-4 energy.`

[nonbond] `Calculate nonbonded energy (electrostatics and van der Waals).`

[elec] `Calculate electrostatic energy.`

[vdw] `Calculate van der Waals energy.`

[etype <type>] `Calculate electrostatics via specified type.`

[simple] `Use simple Coulomb term for electrostatics, no cutoff.`

[directsum] `Use direct summation method for electrostatics.`

- **[npoints <N>]** `Number of cells in each direction to calculate the direct sum.`

[ewald] `Use Ewald summation for electrostatics.`

- **cut <cutoff>** `Direct space cutoff in Angstroms (default 8.0).`
- **dsumtol <dtol>** `Direct sum tolerance (default 0.00001). Used to determine Ewald coefficient.`
- **rsumtol <rtol>** `Reciprocal sum tolerance (default 0.00005). Used to determine number of reciprocal space vectors.`
- **ewcoeff <coeff>** `Ewald coefficient in 1/Ang.`

skinnb `Used to determine pairlist atoms (added to cut, so pairlist cutoff is cut + skinnb); included in order to maintain consistency with results from sander.`

mlimits <X>,<Y>,<Z> `Explicitly set the number of reciprocal space vectors in each dimension.`

`DataSet Aspects:`

[bond] `Bond energy.`

[angle] `Angle energy.`

[dih] `Dihedral energy.`

[vdw14] `1-4 van der Waals energy.`

[elec14] `1-4 electrostatic energy.`

[vdw] `van der Waals energy.`

[elec] `Electrostatic energy.`

[total] `Total energy.`

Calculate the energy for atoms in <mask>. If no terms are specified, all terms are calculated. Note that the non-bonded energy terms for **'simple'** do not take into account periodicity and there is no distance cut-off. Electrostatics can also be determined via the direct sum or Ewald summation (**not** particle mesh) procedures.

This requires that the associated topology file have parameters for any of the calculated terms.

For example, to calculate all energy terms and write to a Grace-format file:

```
parm DPDP.parm7
trajin DPDP.nc
energy DPDP out ene.agr
```

28.9.29. esander

```
esander [<name>] [out <filename>] [saveforces] [parmname <file>] [<namelist vars> ...]
```

[<name>] `Data set name.`

[out <filename>] `File to write results to.`

[saveforces] `If specified, save forces to frames.  Requires writing frames in NetCDF format.`

[parmname <file>] `Name of temporary topology file (default: 'CpptrajEsander.parm7').`

[<namelist vars>] `Namelist variables supported by the sander API in format 'var <value>'; see below.`

Calculate energies for input frames using the sander API. This can be considered as a faster alternative to energy post-processing with sander (imin = 5). Currently the following sander namelist variables are supported: extdiel, intdiel, rgbmax, saltcon, cut, dielc, igb, alpb, gbsa, lj1264, ipb, inp, vdwmeth, ew_type, ntb, ntf, ntc.

If ntb/cut/igb are not specified cpptraj will attempt to pick reasonable values based on the input system. The defaults for a non-periodic system are ntb=0, cut=9999.0, igb=1. The defaults for a periodic system are ntb=1, cut=8.0, igb=0.

For example, to calculate energies for a non-periodic system using igb=1 (the default) with GB surface area turned on (gbsa=1):

```
parm DPDP.parm7
trajin DPDP.nc
esander DPDP out Edpdp.dat gbsa 1
```

28.9.30. filter

```
filter <dataset1 arg> min <min1> max <max1>
       [<dataset2 arg> min <min2> max <max2> ...]
       [out <file> [name <setname>]]
```

<datasetX arg> `Data set name(s) to use for filtering`

min <minX> `Allow values greater than <min> in dataset X.`

max <maxX> `Allow values greater than <max> in dataset X.`

[out <file>] `File containing 1 for frames that were allowed, 0 for frames that were filtered.`

[name <setname>] `Filtered data set name containing 1 for allowed frames, 0 for filtered frames.`

For all following actions, only include frames that are between **<min>** and **<max>** of data sets in **<dataset arg>**. There must be at least one **<min>** and **<max>** argument, and there must be as many **<min>/<max>** arguments as there are specified data sets. For example, to write only frames in-between an RMSD of 0.7-0.8 Angstroms for a given input trajectory:

```
trajin ../tz2.truncoct.nc
rms R1 first :2-11
filter R1 min 0.7 max 0.8 out filter.dat
outtraj maxmin.crd
```

The output trajectory will only contain frames that meet the RMSD requirement, and the *filter.dat* file can be used to see which frames those were that were output.

A similar command that can be used with data that already exists (e.g. it has been read in with ***readdata***) is ***datafilter*** (see page 570).

28.9.31. fixatomorder

```
fixatomorder [outprefix <name>]
```

Cpptraj (and most of Amber) expects that atom indices in molecules to increase monotonically. However, occasionally atom indices in molecules can become disordered. This command fixes atom ordering so that all atoms in molecules are sequential. The **outprefix** keyword will write out the re-ordered topology with name **<name>.<original name>**.

28.9.32. gist (Grid Inhomogeneous Solvation Theory)

```
gist [doorder] [doeij] [skipE] [refdens <rdval>] [temp <tval>]
     [noimage] [gridcntr <xval> <yval> <zval>]
     [griddim <xval> <yval> <zval>] [gridspacn <spaceval>]
     [prefix <filename prefix>] [ext <grid extension>] [out <output>]
     [info <info>]
```

[doorder] `Calculate the water order parameter [557] for each voxel.`

[doeij] `Calculate the triangular matrix representing the water-water interactions between pairs of voxels (see below).`

[skipE] `Skip all energy calculations (cannot be specified with 'doeij').`

[refdens rdval>] `Reference density of bulk water, used in computing g_O, g_H, and the translational entropy.` `Default is 0.0334 molecules/`$Å^3$.

[temp <tval>] `Temperature of the input trajectory.`

[noimage] `Disable distance imaging in energy calculation.`

[gridcntr <xval> <yval> <zval>] `Coordinates (Å) of the center of the grid (default 0.0, 0.0, 0.0).`

[griddim <xval> <yval> <zval>] `Grid dimensions along each coordinate axis (default 40, 40, 40).`

[gridspacn <spaceval>] `Grid spacing (linear dimension of each voxel) in Angstroms. Values greater than 0.75 Å are not recommended (default 0.5 Å).`

[prefix <filename prefix>] `Output file name prefix (default "gist").`

[ext <grid extension>] `Output grid file name extension (default ".dx").`

[out <output>] `Name of the main GIST output file. If not specified set to '<prefix>-output.dat'.`

[info <info>] `Name of main GIST info file. If not specified info is written to standard output.`

`DataSet Aspects:`

[gO] `Number density of oxygen centers found in the voxel, in units of the bulk density.`

[gH] `Number density of hydrogen centers found in the voxel in units of the reference bulk density.`

[Esw] `Mean solute-water interaction energy density.`

[Eww] `Mean water-water interaction energy density.`

[dTStrans] `First order translational entropy density.`

[dTSorient] `First order orientational entropy density .`

[neighbor] `Mean number of waters neighboring the water molecules found in this voxel multiplied by the voxel number density.`

[dipole] `Magnitude of mean dipole moment (polarization).`

[order] `Average Tetrahedral Order Parameter.`

[dipolex] `x-component of the mean water dipole moment density`

[dipoley] `y-component of the mean water dipole moment density`

[dipolez] `z-component of the mean water dipole moment density`

[Eij] `Water-water interaction matrix.`

Grid Inhomogeneous Solvation Theory [558, 559] (GIST) is a method for analyzing the structure and thermodynamics of solvent in the vicinity of a solute molecule. The current implementation works for only water, but the method can be generalized to other solvents whose molecules are rigid like water, such as chloroform or dimethylsulfoxide (DMSO). GIST post-processes explicit solvent simulation data to create a three-dimensional mapping of water density and thermodynamic properties within a region of interest, which is defined by a user-specified 3D rectangular grid. The small grid boxes are referred to as voxels, and each voxel is associated with solvent properties. (See Fig. 28.1.) The GIST implementation incorporated into AmberTools *cpptraj* also calculates a number of other local water properties, as listed below. GIST works for the nonpolarizable water models currently supported by AMBER.

In order to carry out a GIST calculation, you must have a trajectory file generated with explicit water, as well as the corresponding topology file. To generate the most readily interpretable results, it is recommended that the solute (e.g., a protein) be restrained into essentially one conformation. GIST will then provide information about the structure and thermodynamics of the solvent for that conformation. For a room-temperature simulation of a solvent-exposed binding site, and a grid-spacing of 0.5 Å, it is recommended that the simulation be at least 10-20 ns in duration, and it is also a good idea to check for convergence of the GIST properties you are interested in by

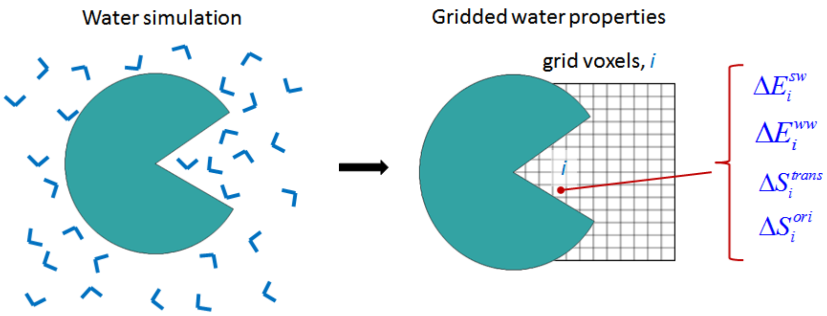


Figure 28.1.: *Diagram, in 2D, of GIST's gridded water properties in a binding site.*

loading and then processing successively more frames of your trajectory file. Because GIST assumes that the solute of interest comprises all molecules in the simulation that are not waters, it is a good idea to remove all counterions and cosolutes with cpptraj's strip command before running GIST. A sample series of cpptraj commands for running GIST is provided below.

Although it is not mandatory to supply values of **gridcntr**, **griddim** and **gridspcn**, these parameters should be carefully chosen, because they determine the region to be analyzed (**gridcntr** and **griddim**) and the spatial resolution and convergence properties of the results (**gridspcn**). In particular, although smaller grid spacings will give finer spatial resolution, longer simulation times will be needed to converge the properties in the smaller voxels that result. A larger grid spacing will allow earlier convergence, but will smooth the spatial distributions and hence can reduce accuracy.

The reference density of water (**rdval**) is taken by default to be the experimental number density of pure water at 300 K and 1 atm. However, different water models may yield slightly different bulk densities under these conditions, and the density also depends on T and P. If you know that the bulk density of the water model you are using, at the T and P of your simulation, deviates significantly from 0.0334 water molecules/Å^3, it would be advisable to supply the actual value with the **refdens** keyword, instead of allowing GIST to supply the default value.

GIST Output

GIST generates a main output file and a collection of grid data files that by default are in Data Explorer format (.dx); this can be changed via the **ext** keyword. These grid files enable visualization of the various gridded quantities, such as with the program VMD [560]. If the **doeij** keyword is provided, GIST also writes out a matrix of water-water interactions between pairs of voxels. In addition, run details are written to stdout, which can be redirected into a log file.

Note that a number of quantities are written out as both densities and normalized quantities. For example, the output file includes both the solute-water energy density and the normalized (per water) solute-water energy. In all cases, the normalized quantity at voxel i, $X_{i,norm}$ is related to the corresponding density, $X_{i,dens}$, by the relationship $X_{i,norm} = \rho_i X_{i,dens}$, where ρ_i is the number density of water in the voxel. The normalized quantity provides information regarding the nature of the water found in the voxel. The density has the property that, if the grid extended over the entire simulation volume, the total system quantity would be given by $X_{tot} = V_{voxel} \sum_i X_{i,dens}$, where V_{voxel} is the volume of one grid voxel.

The main output file takes the form of a space-delimited-variable file, where each row corresponds to one voxel of the grid. This file can easily be opened with and manipulated with spreadsheet programs like Excel and LibreOffice Calc. The columns are as follows.

- **index** - A unique, sequential integer assigned to each voxel

- **xcoord** - x coordinate of the center of the voxel (Å)
- **ycoord** - y coordinate of the center of the voxel (Å)
- **zcoord** - z coordinate of the center of the voxel (Å)
- **population** - Number of water molecule, n_i, found in the voxel over the entire simulation. A water molecule is deemed to populate a voxel if its oxygen coordinates are inside the voxel. The expectation value of this quantity increases in proportion to the length of the simulation.
- **g_O** - Number density of oxygen centers found in the voxel, in units of the bulk density (rdval). Thus, the expectation value of **g_O** for a neat water system is unity.
- **g_H** - Number density of hydrogen centers found in the voxel in units of the reference bulk density (2×rdval). Thus, the expectation value of **g_H** for a neat water system would be unity.
- **dTStrans-dens** - First order translational entropy density (kcal/mole/$Å^3$), referenced to the translational entropy of bulk water, based on the value rdval.
- **dTStrans-norm** - First order translational entropy per water molecule (kcal/mole/molecule), referenced to the translational entropy of bulk water, based on the value rdval. The quantity **dTStrans-norm** equals **dTStrans-dens** divided by the number density of the voxel.
- **dTSorient-dens** - First order orientational entropy density (kcal/mole/$Å^3$), referenced to bulk solvent (see below).
- **dTSorient-norm** - First order orientational entropy per water molecule (kcal/mole/water), referenced to bulk solvent (see below). This quantity equals **dTSorient-dens** divided by the number density of the voxel.
- **Esw-dens** - Mean solute-water interaction energy density (kcal/mole/$Å^3$). This is the interaction of the solvent in a given voxel with the entire solute. Both Lennard-Jones and electrostatic interactions are computed without any cutoff, within the minimum image convention but without Ewald summation. This quantity is referenced to bulk, in the trivial sense that the solute-solvent interaction energy is zero in bulk.
- **Esw-norm** - Mean solute-water interaction energy per water molecule. This equals **Esw-dens** divided by the number density of the voxel (kcal/mole/molecule).
- **Eww-dens** - Mean water-water interaction energy density, scaled by ½ to prevent double-counting, and not referenced to the corresponding bulk value of this quantity (see below). This quantity is one half of the mean interaction energy of the water in a given voxel with all other waters in the system, both on and off the GIST grid, divided by the volume of the voxel (kcal/mole/$Å^3$). Again, both Lennard-Jones and electrostatic interactions are computed without any cutoff, within the minimum image convention.
- **Eww-norm** - Mean water-water interaction energy, normalized to the mean number of water molecules in the voxel (kcal/mole/water). See prior column definition for details.
- **Dipole_x-dens** - x-component of the mean water dipole moment density (Debye/$Å^3$).
- **Dipole_y-dens** - y-component of the mean water dipole moment density (Debye/$Å^3$).
- **Dipole_z-dens** - z-component of the mean water dipole moment density (Debye/$Å^3$).
- **Dipole-dens** - Magnitude of mean dipole moment (polarization) (Debye/$Å^3$).
- **Neighbor-dens** - Mean number of waters neighboring the water molecules found in this voxel multiplied by the voxel number density. Two waters are considered neighbors if their oxygens are within 3.5 angstroms of each other. For any given frame, the contribution to the average is set to zero if no water is found in the voxel (units of number/$Å^3$).

- **Neighbor-norm** - Mean number of neighboring water molecules, per water molecule found in the voxel (units of number per water).
- **Order-norm** - Average Tetrahedral Order Parameter [557], q_{tet}, for water molecules found in the voxel, normalized by the number of waters in the voxel. The order parameter for water i in a given frame is given by: $q_{tet}(i) = 1 - \frac{3}{8}\sum_{j=1}^{3}\sum_{k=j+1}^{4}(cos\phi_{ijk} + \frac{1}{3})^2$ where j and k index the 4 closest water neighbors to water i, and ϕ_{ijk} is the angle formed by water i, j, and k. If the doorder keyword is not provided or is set to FALSE, then this calculation will not be done, and the entries in this column will be set to zero.

Grid files are provided for all computed quantities listed above, except that the normalized quantities are not included. The filenames are as follows: gist-gO.dx, gist-gH.dx, gist-dTStrans-dens.dx, gist-dTSorient-dens.dx, gist-Esw-dens.dx, gist-Eww-dens.dx, gist-dipolex-dens.dx, gist-dipoley-dens.dx, gist-dipolez-dens.dx, gist-dipole-dens.dx, gist-neighbor-dens.dx, gist-neighbor-norm.dx, gist-order-norm.dx. If the **doorder** keyword is not provided, then the data in gist-order-norm.dx will all be zeroes. Note that the file of voxel water densities, gist-gO.dx, can be used as input to the program Placevent [561], in order to define spherical hydration sites based on the density distribution.

Similar grid files with other computed quantities can be generated by reading the gist.out file into a spreadsheet program, processing the numbers to generate a new column of voxel data of interest, and writing this column to an ascii text file. Then the Perl script write_dx_file.pl, which should be available on the GIST tutorial web-site, may be used to read in the column of data and create the corresponding dx file. The input format, and an example, are as follows:

```
./write_dx_file.pl [filename] [x-dimension y-dimension z-dimension]
[x-origin y-origin z-origin] [grid spacing]
./write_dx_file.pl file.dat 40 40 40 13.0 13.0 13.0 0.75
```

If the doeij keyword is provided, GIST also writes a large file, Eww_ij.dat, containing the mean water-water interaction energies between pairs of voxels, scaled by ½. (See below.) This file has three columns. The first two columns are voxel indexes, i, j, where $j > i$, so that no pair appears more than once, and the third column is the mean interaction energy (kcal/mole) of water in voxels i and$j$, scaled by ½. If the occupancy of either voxel is 0, such as for voxels covered by solute atoms, then the interaction energy is zero. In order to save space, such interactions are omitted from the file.

Sample cpptraj input file to run GIST

The following input file, gist.in, causes cpptraj to read a parameter file named topology.top; read in the first 5000 frames of the trajectory file named trajectoryfile.mdcrd; strip out all Na and Cl ions; and carry out a GIST run which computes order parameters, uses a 41x41x45 grid centered at (25.0, 31.0, 30.0) with a spacing of 0.5 Å, uses the default bulk water density of 0.0334 molecules/Å^3, and generates the main output file gist.out.

```
parm topology.top
trajin trajectoryfile.mdcrd 1 5000
strip @Na
strip @Cl
gist doorder doeij gridcntr 25.0 31.0 30.0 griddim 41 41 45
    gridspacn 0.50 out gist.out
go
```

To execute this run in the background, use

```
cpptraj<gist.in>gist.log& or cpptraj -i gist.in>gist.log&
```

Referencing GIST results to unperturbed (bulk) water

Water Model	Mean Energy (Eww-norm) (kcal/mol/water)	Number Density ($Å^{-3}$)
TIP3P	-9.533	0.0329
TIP4PEW	-11.036	0.0332
TIP4P	-9.856	0.0332
TIP5P	-9.596	0.0329
Tip3PFW	-11.369	0.0334
SPCE	-11.123	0.0333
SPCFW	-11.873	0.0329

Table 28.3.: *Water model energy and density.*

Inhomogeneous fluid solvation theory, which is the basis of GIST, is designed to provide information on how water structure and thermodynamics around a solute molecule, such as a protein, are changed relative to the structure and thermodynamics of unperturbed (bulk) water. Accordingly, the quantities reported by GIST are most informative when the results are referenced to the corresponding bulk water properties. For the orientational entropy, the reference value is the same regardless of water model or conditions, because the first order orientational distribution of water in the bulk is always uniform. Therefore, the GIST results for orientational entropies are already referenced to bulk. However, cpptraj reports unreferenced values for those GIST quantities whose reference values depend upon the water model and the simulation conditions; i.e., the energies. The translational entropy as well as the number densities will be referenced to bulk using the input referenced density or the default density value of 0.0334. The table below provides useful reference values for these quantities, computed for various water models at P=1atm, T=300K, using GIST in order to ensure a consistent minimum image treatment of periodic boundary conditions.

Users running calculations under significantly different conditions, or with different water models, should consider generating their own reference quantities by applying GIST to a simulation of pure water under their conditions of interest. The quantities of interest can then be obtained in their most precise available form by averaging over voxels, for the pure water simulation. If the quantity of interest is Q, then its average reference value is $Q_{reference} = \frac{\sum n_i Q_i}{\sum n_i}$, where Q_i and n_i are, respectively, GIST's reported values of the quantity and the population in voxel i. The densities, ρ_i, are referenced to the corresponding bulk densities, ρ^o, as $g_i = \rho_i / \rho^o$, while the energy and entropy terms are referenced by subtracting their bulk values.

Interpreting GIST results

GIST provides access to the first order entropies and the first- and second-order energies of inhomogeneous fluid solvation theory. Non-zero higher-order entropies exist but are not yet computationally accessible. However, for a pairwise additive force-field, such as those listed in the Table above, the energy is fully described at the second order provided by GIST.

GIST is a research tool, and its applications (to, for example, protein-ligand binding and protein function) are still being explored. The following general comments may be helpful to users studying GIST results.

1. The water in voxels near a solute (e.g., a protein) almost always has unfavorable water-water interaction energies, relative to bulk, simply because the solute displaces water, resulting in fewer proximal water-water interactions.

2. The unfavorable water-water energies mentioned in [558] may be balanced by favorable water-solute interactions. If they are not, as may occur especially for voxels in small, hydrophobic pockets, then the net energy of the water in the voxel may be unfavorable relative to bulk, in which case a ligand which displaces water from the voxel into bulk may get a boost in affinity.

3. Because the first order orientational distribution of bulk water is uniform, and a nonuniform distribution always has lower entropy than a uniform one, the solute can only lower the orientational entropy of water, relative to bulk. Thus, this term always opposes solvation, and displacing oriented water into the bulk is always favorable from the standpoint of orientational entropy.

4. Localized water, which corresponds to voxels with high water density, has a low first order translational entropy, and the translational entropy around a solute is lower than that in bulk, as a nonuniform translational distribution takes the place of the uniform translational distribution of bulk water.

5. The displacement of highly oriented (low orientational entropy) and localized (low translational entropy) water into bulk leads to a favorable increase in these entropy terms.

6. However, highly oriented and localized water is often the consequence of strongly favorable polar interactions, such as hydrogen-bonding, between water and the solute. As a consequence, the net favorability of displacing such water is frequently a balance between favorable entropic consequences and unfavorable energetic consequences.

7. The water-water energy associated with a given voxel accounts for the interactions of the waters in this voxel with all other waters in the system, including waters in other voxels. This quantity is multiplied by ½, so that, in a pure-water system where the GIST grid covers the entire simulation box, the sum over all voxels equals the correct mean water-water interaction energy. Note that Reference [559] does not include this factor of ½.

8. For a typical GIST application, in which the grid occupies only part of the simulation box, the energy bookkeeping can become complicated, as discussed in Section II.B.3 (page 044101-6) of Reference [559]. That section also explains how one can compute the water-water energy associated with a region R defined by a set of voxels, E^R_{WW}. The regional water-water energy, on a normalized (per water) basis, is given by $E^R_{WW} = 2(\sum_{i\in R} E_{i,WW} - \sum_{i\in R}\sum_{j\in R,j>i} E_{i,j,WW})$ where $i \in R$ means that voxel i is in region R, $E_{i,WW}$ is the value of Eww-norm for voxel i, and $E_{i,j,WW}$ is the value of the water-water interaction energy between voxels i and j, taken from the file Eww_ij.dat. The extra factor of 2 in the present formula, relative to that in the paper, results from application of an extra factor of ½ to the reported water-water interaction energies here.

28.9.33. grid

```
grid <filename>
  { data <dsname> | boxref <ref name/tag> <nx> <ny> <nz> |
    <nx> <dx> <ny> <dy> <nz> <dz> [gridcenter <cx> <cy> <cz>] }
  [box|origin|center <mask>] [negative] [name <gridname>]
  <mask> [normframe | normdensity [density <density>]]
 [pdb <pdbout> [max <fraction>]]
  [[smoothdensity <value>] [invert]] [madura <madura>]
```

<filename> File to write out grid to. Use ".grid" or ".xplor" extension for XPLOR format, ".dx" for OpenDX format.

Options for setting up grid:

data <dsname> Use previously calculated/loaded grid data set named <dsname>. When using this option there is no need to specify grid bins/spacing/center.

boxref <ref name/tag> <nx> <ny> <nz> Set up grid using box information from a previously loaded reference structure. Currently the only way to set up non-orthogonal grids.

<nx> <dx> <ny> <dy> <nz> <dz> Number of grid bins and spacing in the X/Y/Z directions.

[gridcenter <cx> <cy> <cz>] Location of grid center, default is origin (0.0, 0.0, 0.0).

Options for offset during grid binning (must center grid at origin):

[box] Offset each point by location of box center prior to gridding. Cannot be used with 'gridcenter'.

[origin] No offset (default)

[center <mask>] Offset each point by center of atoms in <mask> prior to gridding. Cannot be used with 'gridcenter'.

Other options:

[negative] Grid negative density instead of positive density.

[name <gridname>] Grid data set name.

<mask> Mask of atoms to grid.

[normframe] Normalize grid bins by the number of frames.

[normdensity [density <density>]] Normalize grid bins by density: GridBin = GridBin / (Nframes * BinVolume * density). Default particle density (molecules/Ang^3) for water based on 1.0 g/mL.

[pdb <pdbout> [max <fraction>]] Write a pseudo-PDB of grid points that have density greater than <fraction> (default 0.80) of the grid max value.

Less common options:

[smoothdensity <smooth>] Used to smooth density. The smoothing takes the form of GridBin = 0 if GridBin < smooth, otherwise GridBin = GridBin - (GridBin * exp[-(GridBin - smooth)^2 / (0.2 * smooth^2)]).

[invert] (Only used if smoothdensity also used) Do inverse smoothing (i.e. if GridBin > smooth).

[madura <madura>] Grid values lower than <madura> become flipped in sign, exposes low density.

Data Sets Created:

<dsname> Grid data set.

Create a grid representing the histogram of atoms in *mask1* on the 3D grid that is "*nx* * *x_spacing* by *ny* * *y_spacing* by *nz* * *z_spacing* angstroms (cubed). By default the grid is centered at the origin unless **gridcenter** is specified. Grid points can be offset by either the box center (using **box**) or the center of specified atoms (using **center <mask>**); if either of these options are used the grid must be centered at the origin. Note that the ***bounds*** command (on page 594) can be very useful for determining grid dimensions.

Note that when calculating grid densities for things like solvent/ions, the solute of interest (about which the atomic densities are binned) should be rms fit, centered and imaged prior to the ***grid*** call in order to provide any meaningful representation of the density. If the optional keyword `negative` is also specified, then these density will be stored as negative numbers. Output can be in the XPLOR or OpenDX data formats.

Examples

Grid water density around a solute.

```
trajin tz2.truncoct.nc
autoimage origin
rms first :1-13
# Create average of solute to view with grid.
average avg.mol2 :1-13
grid out.dx 20 0.5 20 0.5 20 0.5 :WAT@O
```

Generate grid from bounds command.

```
trajin tz2.ortho.nc
autoimage
rms first :1-13&!@H= mass
bounds :1-13 dx .5 name MyGrid out bounds.dat
average bounds.mol2 :1-13
# Save coordinates for second pass.
createcrd MyCoords
```

```
run
# Grid using grid data set from bounds command.
crdaction MyCoords grid bounds.xplor data MyGrid :WAT@O
```

Create non-orthogonal grid:

```
trajin tz2.truncoct.nc
reference ../tz2.truncoct.nc [REF]
autoimage triclinic
grid nonortho.dx boxref [REF] 50 50 50 :WAT@O pdb nonortho.pdb
```

28.9.34. hbond

```
hbond [<dsname>] [out <filename>] [<mask>] [angle <acut>] [dist <dcut>]
      [donormask <dmask> [donorhmask <dhmask>]] [acceptormask <amask>]
      [avgout <filename>] [printatomnum] [nointramol] [image]
      [solventdonor <sdmask>] [solventacceptor <samask>]
      [solvout <filename>] [bridgeout <filename>]
      [series [uuseries <filename>] [uvseries <filename>]]
```

[<dsname>] `Data set name.`

[out <filename>] `Write # of solute-solute hydrogen bonds (aspect [UU]) vs time to this file. If searching for solute-solvent hydrogen bonds, write # of solute-solvent hydrogen bonds (aspect [UV]) and # of bridging solvent molecules (aspect [Bridge]), as well as the residue # of the bridging solvent and the solute residues being bridged with format '<solvent resnum>(<solute res1>+<solute res2>+...+),...' (aspect [ID]).`

[<mask>] `Atoms to search for solute hydrogen bond donors/acceptors.`

[angle <acut>] `Angle cutoff for hydrogen bonds (default 135°). Can be disabled by specifying -1.`

[dist <dcut>] `Distance cutoff for hydrogen bonds (acceptor to donor heavy atom, default 3.0 Å).`

[donormask <dmask>] `Use atoms in <dmask> as solute donor heavy atoms. If 'donorhmask' not specified only atoms bonded to hydrogen will be considered donors.`

[donorhmask <dhmask>] `Use atoms in <dmask> as solute donor hydrogen atoms. Should only be specified if 'donormask' is. Should be a 1 to 1 correspondence between donormask and donorhmask.`

[acceptormask <amask>] `Use atoms in <amask> as solute acceptor atoms.`

[avgout <filename>] `Write solute-solute hydrogen bond averages to <filename>.`

[printatomnum] `Add atom numbers to the output, in addtion to residue name, residue number and atom name.`

[nointramol] `Ignore intramolecular hydrogen bonds.`

[image] `Turn on imaging of distances/angles.`

[solventdonor <sdmask>] `Use atoms in <sdmask> as solvent donors. Can specify ions as well.`

[solventacceptor <samask>] `Use atoms in <samask> as solvent acceptors. Can specify ions as well.`

[solvout <filename>] `Write solute-solvent hydrogen bond averages to <filename>. If not specified and 'avgout' is, solute-solvent hydrogen bonds averages will be written to that file.`

[bridgeout <filename>] `Write information on detected solvent bridges to <filename>. If not specified, will be written to same place as 'solvout'.`

[series] `Save hydrogen bond formed (1.0) or not formed (0.0) per frame for any detected hydrogen bond. Solute-solute hydrogen bonds are saved with aspect [solutehb], solute-solvent hydrogen bonds are saved with aspect [solventhb].`

  [uuseries <filename>] `File to write solute-solute hbond time series data to.`

  [uvseries <filename>] `File to write solute-solvent hbond time series data to.`

`Data Sets Created:`

<dsname>[UU] `Number of solute-solute hydrogen bonds.`

<dsname>[UV] `(only for solventdonor/solventacceptor) Number of solute-solvent hydrogen bonds.`

<dsname>[Bridge] `(only for solventdonor/solventacceptor) Number of bridging solvent molecules.`

<dsname>[ID] `(only for solventdonor/solventacceptor) String identifying bridging solvent residues and the solute residues they bridge.`

<dsname>[solutehb] `(series only) Time series for solute-solute hydrogen bonds; 1 for present, 0 for not present.`

<dsname>[solventhb] `(series only) Time series for solute-solvent hydrogen bonds; 1 for present, 0 for not present.`

*Note that **series** data sets are not generated until hydrogen bonds are actually determined (i.e. **run** is called).*

Determine hydrogen bonds in each coordinate frame using simple geometric criteria. A hydrogen bond is defined as being between an acceptor heavy atom A, a donor hydrogen atom H, and a donor heavy atom D. If the A to D distance is less than the distance cutoff and the A-H-D angle is greater than the angle cutoff a hydrogen bond is considered formed. Imaging of distances/angles is not performed by default, but can be turned on using the **image** keyword.

Potential hydrogen bond donor/acceptor atoms are searched for as follows:

1. If just **<mask>** is specified donors and acceptors will be automatically determined from **<mask>**.
2. If **donormask** is specified donors will be determined from **<dmask>** (only atoms bonded to hydrogen will be considered valid). Optionally, **donorhmask** can be used in conjunction with **donormask** to explicitly specify the hydrogen atoms bonded to donor atoms. Acceptors will be automatically determined from **<mask>**.
3. If **acceptormask** is specified acceptors will be determined from **<amask>**. Donors will be automatically determined from **<mask>**.
4. If both **acceptormask** and **donormask** are specified only **<amask>** and **<dmask>** will be used; no searching will occur in **<mask>**.

Automatic determination of hydrogen bond donors/acceptors uses the simplistic criterion that “hydrogen bonds are FON”, i.e., hydrogens bonded to F, O, and N atoms are considered donors, and F, O, and N atoms are considered acceptors. Intra-molecular hydrogen bonds can be ignored using the **nointramol** keyword.

The number of hydrogen bonds present at each frame will be determined and written to the file specified by **out**. If the **series** keyword is specified the time series for each hydrogen bond (1 for present, 0 for not present) will also be saved for subsequent analysis (e.g. with ***lifetime***, see on page 665); solute-solute hydrogen bonds will be saved to '<dataset name>[solutehb]' and solute-solvent hydrogen bonds will be saved to '<dataset name>[solventhb]'. The data set legends are set with the residues and atoms involved in the hydrogen bonds. In the case of solute to non-specific solvent hydrogen bonds, a V is used in place of solvent.

If **avgout** is specified the average of each solute-solute hydrogen bond (sorted by population) formed over the course of the trajectory is printed with the format:

```
Acceptor   DonorH   Donor   Frames   Frac   AvgDist   AvgAng
```

where *Acceptor*, *DonorH*, and *Donor* are the residue and atom name of the atoms involved in the hydrogen bond, *Frames* is the number of frames the bond is present, *Frac* is the fraction of frames the bond is present, *AvgDist* is the average distance of the bond when present, and *AvgAng* is the average angle of the bond when present. The **printatomnum** keyword can be used to print atom numbers as well.

Solute to non-specific solvent hydrogen bonds can be tracked by using the **solventdonor** and/or **solventacceptor** keywords. The number of solute-solvent hydrogen bonds and number of "bridging" solvent molecules (i.e. solvent that is hydrogen bonded to two or more different solute residues at the same time) will also be written to the file specified by **out**. These keywords can also be used to track non-specific interactions with ions. If **avgout** or **solvavg** is specified the average of each solute solvent hydrogen bond will be printed with the format:

```
Acceptor   DonorH   Donor   Count   Frac   AvgDist   AvgAng
```

where *Acceptor*, *DonorH*, and *Donor* are either the residue and atom name of the solute atoms or "SolventAcc"/"SolventH"/"SolventDnr" representing solvent, *Count* is the total number of interactions between solute and solvent (note this can be greater than the total number of frames since for any given frame more than one solvent molecule can hydrogen bond to the same place on solute and vice versa), *AvgDist* is the average distance of the bond when present, and *AvgAng* is the average angle of the bond when present. If **avgout** or **bridgeout** is specified information on residues that were bridged by a solvent molecule over the course of the trajectory will be written to <bfilename> with format:

```
Bridge Res <N0:RES0> <N1:RES1> ... , <X> frames.
```

where '<N0:RES0> ...' is a list of residues that were bridged (residue # followed by residue name) and <X> is the number of frames the residues were bridged.

hbond Examples

To search for all hydrogen bonds within residues 1-22, writing the number of hydrogen bonds per frame to "nhb.dat" and information on each hydrogen bond found to "avghb.dat":

```
hbond :1-22 out nhb.dat avgout avghb.dat
```

To search for all hydrogen bonds formed between donors in residue 1 and acceptors in residue 2:

```
hbond donormask :1 acceptormask :2 out nhb.dat avgout avghb.dat
```

To search for all intermolecular hydrogen bonds only and solute-solvent hydrogen bonds, saving time series data to HB:

```
hbond HB out nhb.dat avgout solute_avg.dat \
  solventacceptor :WAT@O solventdonor :WAT \
  solvout solvent_avg.dat bridgeout bridge.dat \
  series uuseries uuhbonds.agr uvseries uvhbonds.agr
```

To search for non-specific hydrogen bonds between solute and ions named Na+:

```
hbond HB-Ion out nhb.agr avgout ion_avg.dat \
  solventacceptor :Na+ solventdonor :Na+
```

28.9.35. image

```
image [origin] [center] [triclinic | familiar [com <commask>]] [<mask>]
      [ bymol | byres | byatom ] [xoffset <x>] [yoffset <y>] [zoffset <z>]
```

[origin] `Image to coordinate origin (0.0, 0.0, 0.0); default is to image to box center.`

[center] `For bymol/byres, image by center of mass; default is to image by first atom position.`

[triclinic] `Force imaging with triclinic code.  This is the default for non-orthorhombic cells.`

[familiar [com <commask>]] `Image to truncated octahedron shape.  If 'com <commask>' is given, image with respect to the center of mass of atoms in <commask>.`

[<mask>] `Image atoms/residues/molecules in mask.`

[bymol] `Image by molecule (default).`

[byres] `Image by residue.`

[byatom] `Image by atom.`

[xoffset <x>] `Shift atoms by a factor of <x> in the X-direction.`

[yoffset <y>] `Shift atoms by a factor of <y> in the Y-direction.`

[zoffset <z>] `Shift atoms by a factor of <z> in the Z-direction.`

Note this command is intended for advanced use; for most cases the ***autoimage*** command should be sufficient.

For periodic systems only, image molecules/residues/atoms that are outside of the box back into the box. Currently both orthorhombic and non-orthorhombic boxes are supported. A typical use of ***image*** is to move molecules back into the box after performing ***center***. For example, the following commands move all atoms so that the center of residue 1 is at the center of the box, then image so that all molecules that are outside the box after centering are wrapped back inside:

```
center :1
image
```

The xoffset etc. keywords can be used to shift the entire unit cell in a certain direction by the given factor, which can be useful for visualizing trajectories with periodic boundary conditions. For example, to generate a trajectory that is offset by 1.0 box length in the X direction, one could use:

```
image xoffset 1.0
trajout traj.offsetx1.nc
```

28.9.36. jcoupling

```
jcoupling <mask> [outfile <filename>] [kfile <param file>] [out <filename>]
          [name <dsname>]
```

<mask> `Atom mask in which to search for dihedrals within.`

[outfile <filename>] `File to write j-coupling values to with fixed format.`

[kfile <param file>] `File containing Karplus parameters (default is $AMBERHOME/dat/Karplus.txt).`

[out <filename>] `File to write data set output to.`

[name <dsname>] `Data set name.`

Note data sets are not generated until ***run*** *is called.*

Calculate J-coupling values for all dihedrals found within **<mask>** (all atoms if no mask given). In order to use this function, Karplus parameters for all dihedrals which will be calculated must be loaded. By default *cpptraj* will use the data found in $AMBERHOME/dat/Karplus.txt; if this is not found *cpptraj* will look for the file specified by the $KARPLUS environment variable.

In the Karplus parameter file each parameter set consists of two lines for each dihedral with the format:

```
[<Type>]<Name1><Name2><Name3><Name4><A><B><C>[<D>]
<Resname1>[<Resname2>...]
```

The first line defines the parameter set for a dihedral. <Type> is optional; if not given the form for calculating the J-coupling will be as described by Chou et al.[562]; if 'C' the form will be as described by Perez et al.[563]. The <NameX> parameters define the four atoms involved in the dihedral. Each <NameX> parameter is 5 characters wide, starting with a plus '+', minus '-' or space ' ' character indicating the atom belongs to the next, previous, or current residue. The remaining 4 characters are the atom name. The parameters <A>, <B>, <C>, and <D> are floating point values 6 characters wide describing the Karplus parameters. For the 'C' form A, B, and C correspond to C0, C1, and C2; D is unused and should not be specified. The second line is a list of residue names (4 characters each) to which the dihedral applies. For example:

```
C HA   CA   CB   HB    5.40 -1.37  3.61
ILE VAL
```

Describes a dihedral between atoms HA-CA-CB-HB using the Perez et al. form with constants C0=5.40, C1=-1.37, C2=3.61 applied to ILE and VAL residues.

Output can be in both a fixed format (**outfile <filename>**) and using cpptraj data set/data file formatting (**out <filename>**). The fixed format has each dihedral that is defined from **<mask1>** printed along with its calculated J-coupling value for each frame, e.g.:

```
#Frame 1
1 SER HA CA CB HB2 45.334742 4.024759
1 SER HA CA CB HB3 -69.437134 1.829510
...
```

First the frame number is printed, then for each dihedral: Residue number, residue name, atom names 1-4 in the dihedral, the value of the dihedral, the J-coupling value.

In `cpptraj` format, only the J-coupling value is written.

28.9.37. lessplit

```
lessplit [out <filename prefix>] [average <avg filename>] <trajout args>
```

[out <filename prefix>] `Write split LES trajectories to <filename prefix>.X, where X is an integer.`

[average <avg filename>] `Write trajectory of averaged LES regions to <avg filename>.`

<trajout args> `Arguments for output trajectories.`

Split and/or average LES trajectory. At least one of **'out'** or **'average'** must be specified. If both are specified they share <trajout args>.

28.9.38. lie

```
lie [<name>] <Ligand mask> [<Surroundings mask>] [out <filename>]
    [noelec] [novdw] [cutvdw <cutoff>] [cutelec <cutoff>] [diel <dielc>]
DataSet Aspects:
```

[EELEC] `Electrostatic energy (kcal/mol).`

[EVDW] `van der Waals energy (kcal/mol).`

For each frame, calculate the non-bonded interactions between all atoms in <Ligand mask> with all atoms in <Surroundings mask>. Electrostatic and van der Waals interactions will be calculated for all atom pairs. A separate electrostatic and van der Waals cutoff can be applied, the default is 12.0 Angstroms for both. <dielc> is an optional dielectric constant. Either the electrostatic or van der Waals calculations can be suppressed via the keywords noelec and novdw, respectively.

The electrostatic interactions are calculated according to a simple shifting function shown below. The data file will contain two data sets—one for electrostatic interactions and one for van der Waals interactions. Periodic topologies and trajectories are required (i.e., explicit solvent is necessary). The minimum image convention is followed.

$$E_{elec} = k\frac{q_i q_j}{r_{ij}}\left(1 - \frac{r_{ij}^2}{r_{cut}^2}\right)^2$$

28.9.39. lipidorder

```
order out <filename> [x|y|z] [scd] [unsat <mask>]
      [taildist <filename> [delta <resolution>] tailstart <mask>
       tailend <mask>] <mask0> ... <maskN>
```

out `Output file for order parameters: Sx, Sy, Sz (each succeeded by the standard deviation), and two estimates for the deuterium-order parameter |SCD| = 0.5Sz and |SCD| = -(2Sx + Sy)/3. If scd is set then the order parameter directly computed from the C-H vectors is output.`

x|y|z `Reference axis. (z)`

unsat `Mask for unsaturated bonds. Sz is calculated for vector Cn-Cn+1. This is only relevant if scd (below) is not set, i.e. order parameters are calculated from carbon position only.`

scd `Calculate the deuterium-order parameter |SCD| directly from the C-H vectors (masks must contain C-H-H triplets, see below). Otherwise the order parameter is estimated from carbon positions only (masks must contain only relevant carbons). (false)`

taildist `Optional output file for end-to-end distances.`

delta `Optional resolution for taildist. (0.1)`

tailstart `Mask for the start of the tail. Must be given if taildist.`

tailend `Mask for the end of the tail. Must be given if taildist.`

mask0 `... maskN Masks for each group in the lipid chain.`

The order parameters S_x, S_y, S_z and |SCD| are calculated. Carbons must be given in bonding order. If **scd** the masks must be made up of C-H-H triples, hence hydrogens to double bonds must be enumerated twice while methyl groups require an additional mask which will also create two entries in the output.

$S_z$is the vector joining carbons C_{n-1} and C_{n+1}, S_x the vector normal to the $C_{n-1} - C_n$ and $C_n - C_{n+1}$ plane and S_y is the third axis in the molecular coordinate system. The order parameter is then calculated from $Sc = 0.5 < 3\cos(2\theta) > -1$, where θ is the angle to the chosen reference axis. See example input file.

Example input (all atom names according to CHARMM27 force field for POPC).

sn1 chain: order parameters S_x, S_y, S_z and $|SCD| = 0.5 \times S_z$ and $|SCD| = -(2S_x + S_y)/3$

```
lipidorder out sn1.dat z taildist e2e_sn1.dat delta 0.1 \
           tailstart ":POPC@C32" tailend ":POPC@C316" \
           ":POPC@C32" ":POPC@C33" ":POPC@C34" ":POPC@C35" \
           ":POPC@C36" ":POPC@C37" ":POPC@C38" ":POPC@C39" \
           ":POPC@C310" ":POPC@C311" ":POPC@C312" ":POPC@C313" \
           ":POPC@C314" ":POPC@C315" ":POPC@C316"
```

See also `$AMBERHOME/AmberTools/test/cpptraj/Test_LipidOrder`.

28.9.40. makestructure

```
makestructure <List of Args>
```

Apply dihedrals to specified residues using arguments found in <List of Args>, where an argument is 1 or more of the following arg types:

<sstype keyword>:<res range>

Apply secondary structure type (via phi/psi backbone angles) to residues in given range. If the secondary structure type is a turn, the residue range must correspond to a multiple of 2 residues.

Keyword	phi, psi (deg.)	# residues
alpha	-57.8, -47.0	1
left	-57.8, 47.0	1
pp2	-75.0, 145.0	1
hairpin	-100.0, 130.0	1
extended	-150.0, 155.0	1
typeI	-60.0, -30.0 | -90.0, 0.0	2
typeII	-60.0, 120.0 | 80.0, 0.0	2
typeVIII	-60.0, -30.0 | -120.0, 120.0	2
typeI’	60.0, 30.0 | 90.0, 0.0	2
typeII	60.0, -120.0 | -80.0, 0.0	2
typeVIa1	-60.0, 120.0 | -90.0, 0.0	2
typeVIa2	-120.0, 120.0 | -60.0, 0.0	2
typeVIb	-135.0, 135.0 | -75.0, 160.0	2

<custom ss>:<res range>[:<phi>:<psi>]

If <phi> and <psi> are given, define a custom secondary structure conformation named <custom_ss> and apply to residues in range. If <custom_ss> has been previously defined then apply it to residues in range.

<custom turn>:<res range>[:<phi1>:<psi1>:<phi2>:<psi2>]

If <phi1>, <psi1>, <phi2>, and <psi2> are given, defined a custom turn conformation named <custom_turn> and apply to residues in range (range must correspond to a multiple of 2 residues). If <custom_turn> has been previously defined then apply it to residues in range.

<custom dih>:<res range>[:<dih type>:<angle>]

```
<dih type> = phi psi chip omega alpha beta gamma delta epsilon zeta nu1 nu2 chin
```

If <dih type> and <angle> are given, apply <angle> to selected dihedrals of type in range. If <custom dih> has been previously defined then apply it to residues in range.

<custom dih>:<res range>[:<at0>:<at1>:<at2>:<at3>:<angle>[:<offset>]]

Apply <angle> to dihedral defined by atoms <at1>, <at2>, <at3>, and <at4>, or use previously defined <custom_dih>.

<offset>	Description
-2	<at0> and <at1> in previous residue.
-1	<at0> in previous residue.
0	All atoms in single residue.
1	<at3> in next residue.
2	<at2> and <at3> in next residue.

ref:<range>:<refname>[:<ref range>]

Apply dihedrals from residues <ref_range> in previously loaded reference structure <refname> to dihedrals in <range>.

Examples

Assign polyproline II structure to residues 1 through 13:

```
makestructure pp2:1-13
```

Make residues 1 and 12 ’extended’, residues 6 and 7 a type I’ turn, and two custom assignments, one (custom1) for residues 2-5, the other (custom2) for residues 8-11:

```
makestructure extended:1,12 \
              custom1:2-5:-80.0:130.0:-130.0:140.0 \
              typeI':6-7 \
              custom2:8-11:-140.0:170.0:-100.0:140.0
```

Assign residue 5 phi 90 degrees, residues 6 and 7 phi=-70 and psi=60 degrees:

```
makestructure customdih:5:phi:90 custom:6,7:-70:60
```

Create a new dihedral named chi1 and assign it a value of 35 degrees in residue 8:

```
makestructure chi1:8:N:CA:CB:CG:35
```

Assign ’extended’ structure to residues 1 and 12, a custom turn to residues 2-5 and 8-11, and a typeI’ turn to residues 6-7:

```
makestructure extended:1,12 \
              custom1:2-5:-80.0:130.0:-130.0:140.0 \
              typeI':6-7 \
              custom1:8-11
```

Assign secondary structure from reference structure:

```
parm ../tz2.parm7
reference ../tz2.rst7
trajin pp2.rst7.save
makestructure "ref:1-13:tz2.rst7" rmsd reference
trajout fromref.pdb multi
```

28.9.41. mask

```
mask <mask> [maskout <filename>] [maskpdb <pdbname>] [maskmol2 <mol2name>]
```

<mask> `Atom mask to process.`

[maskout <filename>] `Write information on atoms in <mask> to <filename>.`

[maskpdb <name>] `Write PDB of atoms in <mask> to <name>.X.`

[maskmol2 <name>] `Write Mol2 of atoms in <mask> to <name>.X.`

For each frame determine all atoms that correspond to **<mask>**. This is most useful when using distance-based masks, since the atoms in the mask are updated for every frame read in. If **maskout** is specified information on all atoms in <mask> will be written to <filename> with format:

```
#Frame AtomNum Atom ResNum Res MolNum
```

where `#Frame` is the frame number, `AtomNum` is the number of the selected atom, `Atom` is the name of the selected atom, `ResNum` is the residue number of the selected atom, `Res` is the residue name, and `MolNum` is the molecule number of the selected atom.

If **maskpdb** or **maskmol2** are specified a PDB/Mol2 file corresponding to <mask> will be written out every frame with name "*<name>.frame#*".

For example, to write out all atoms within 3.0 Angstroms of residue 195 that are part of residues named WAT to "Res195WAT.dat", as well as write out corresponding PDB files:

```
mask "(:195<:3.0)&:WAT" maskout Res195WAT.dat maskpdb Res195WAT.pdb
```

28.9.42. matrix

```
matrix [out <filename>] [start <#>] [stop|end <#>] [offset <#>]
       [name <name>] [ byatom | byres [mass] | bymask [mass] ]
       [ ired [order <#>] ]
       [ {distcovar | idea} <mask1> ]
       [ {dist | correl | covar | mwcovar} <mask1> [<mask2>] ]
       [ dihcovar dihedrals <dataset arg> ]
```

[out <filename>] `If specified, write matrix to <filename>.`

[start <#>] [stop|end <#>] [offset <#>] `Start, stop, and offset frames to use (as a subset of all frames read in).`

[name <name>] `Name of the matrix dataset (for referral in subsequent analysis).`

byatom `Write results by atom (default). This is the sole option for covar, mwcovar, and ired.`

byres `Write results by calculating an average for each residue (mass weighted if mass is specified).`

bymask `Write average over <mask1>, and if <mask2> is specified <mask1> x <mask2> and <mask2> as well (mass weighted if mass is specified).`

Calculate matrix of the specified type from input coordinate frames:

dist <mask1> [<mask2>] Distance matrix (default).

correl <mask1> [<mask2>] Correlation matrix (aka dynamic cross correlation[564]).

covar <mask1> [<mask2>] Coordinate covariance matrix.

mwcovar <mask1> [<mask2>] Mass-weighted coordinate covariance matrix.

distcovar <mask1> Distance covariance matrix.

idea <mask1> Isotropically Distributed Ensemble Analysis matrix.[565]

ired [order <#>] Isotropic Reorientational Eigenmode Dynamics matrix[566] with Legendre polynomials of specified order (default 1). IRED vectors must have been specified previously with '***vector* ired**' (see 28.9.79 on page 645).

dihcovar dihedrals <dataset arg> Dihedral covariance matrix. Dihedral data sets must have been previously defined with e.g. ***dihedral*** or ***multidihedral*** commands or read in externally with ***readdata*** and marked as dihedrals.

Matrix dimensions will be of the order of N x M for **dist**, **correl**, **idea**, and **ired**, 2N x 2N for **dihcovar**, 3N x 3M for **covar** and **mwcovar**, and N(N-1) x N(N-1) / 4 for **distcovar** (with N being the number of data sets in the case of **ired** and **dihcovar** and the number of atoms in **<mask1>** otherwise, and M being the number of atoms in **<mask2>** if specified or **<mask1>** otherwise). No mask is required for **ired**; the matrix will be made up of previously defined

IRED vectors (see the ***vector*** command on page 645). Similarly no mask is required for dihcovar; dihedral data sets must have been previously defined. Only one mask can be used with **distcovar** and **idea** matrices (i.e. they can be symmetric only), otherwise one or two masks can be used (for symmetric and full matrices respectively). If two masks are specified the number of atoms covered by *mask1* must be greater than or equal to the number of atoms covered by *mask2,* and on output **<mask1>** corresponds to columns while **<mask2>** corresponds to rows.

As a simple example, a distance matrix of all CA atoms is generated and output to distmat.dat.

```
matrix dist @CA out distmat.dat
```

28.9.43. mindist

This functionality is now part of the ***nativecontacts*** command; see 28.9.49 on page 627.

28.9.44. minimage

```
minimage [<name>] <mask1> <mask2> [out <filename>] [geom] [maskcenter]
```

<name> `Data set name.`

<mask1> `First atom mask.`

<mask2> `Second atom mask.`

out <filename> `File to write to.`

geom `(maskcenter only) If specified, use geometric center instead of center of mass.`

maskcenter `Calculate distance from center of masks instead of between each atom.`

`Data Sets Created:`

<name> `Minimum distance to an image in Ang.`

<name>[A1] `Atom number in mask 1 involved in minimum distance.`

<name>[A2] `Atom number in mask 2 involved in minimum distance.`

Calculate the shortest distance to an image, i.e. the distance to a neighboring unit cell, as well as the numbers of the atoms involved in the distance. By default the distance between each atom in <mask1> and <mask2> is considered; if **maskcenter** is specified the center of the masks is used.

28.9.45. molsurf

```
molsurf [<name>] [<mask>] [out filename] [probe <probe_rad>]
     [radii {gb | parse | vdw}] [offset <rad_offset>]
```

[<name>] `Name of surface area data set.`

[<mask>] `Atoms to calculate surface area of.`

[out <filename>] `File to write values to.`

[probe <probe_rad>] `Probe radius (default 1.4 Angstrom).`

[offset <rad_offset>] `Add <rad_offset> to each atom radius (default 0.0).`

[radii {gb|parse|vdw}] `Specify radii to use:`

- **gb** `GB radii (default).`
- **parse** `PARSE radii.`
- **vdw** `van der Waals radii.`

Calculate the Connolly surface area[567] of atoms in <mask> (default all atoms if no mask specified) using routines from molsurf (originally developed by Paul Beroza) using the probe radius specified by **probe** (1.4 Å if not specified). Note that if GB/VDW radii are not present in the topology file (e.g. for PDB files), then PARSE radii can be used. Also note that this routine only calculate absolute surface areas, i.e. it cannot be used to get the contribution of a subset of atoms to overall surface area; if such functionality is needed try the ***surf*** command (28.9.73 on page 642).

28.9.46. multidihedral

```
multidihedral [<name>] <dihedral types> [resrange <range>] [out <filename>] [range360]
              [dihtype <name>:<a0>:<a1>:<a2>:<a3>[:<offset>] ...]
                  Offset -2=<at0><at1> in previous res, -1=<at0> in previous res,
                          0=All <atX> in single res,
                          1=<at3> in next res, 2=<at2><at3> in next res.
              <dihedral types> =  phi psi chip omega alpha beta gamma delta
                                  epsilon zeta nu1 nu2 chin
```

[<name>] `Output data set name.`

<dihedral types> `Dihedral types to look for. Note that chip is 'protein chi', chin is 'nucleic chi'.`

[resrange <range>] `Residue range to look for dihedrals in.`

[out <filename>] `Output file name.`

[range360] `Wrap torsion values from 0.0 to 360.0 (default is -180.0 to 180.0).`

[dihtype <name>:<a0>:<a1>:<a2>:<a3>[:<offset>] `Search for a custom dihedral type called <name> using atom names <a0>, <a1>, <a2>, and <a3>.`

`Offset: -2=<a0><a1> in previous res, -1=<a0> in previous res, 0=All <aX> in single res, 1=<a3> in next res, 2=<a2><a3> in next res.`

`DataSet Aspects:`

[<dihedral type>] `Aspect corresponds to the dihedral type name (e.g. [phi], [psi], etc).`

Note data sets are not generated until ***run*** *is called.*

Calculate specified dihedral angle types for residues in given range. By default, dihedral angles are identified based on standard Amber atom names. The resulting data sets will have aspect equal to [<dihedral type>] and index equal to residue #. To differentiate the chi angle, chip is used for proteins and chin for nucleic acids. For example, to calculate all phi/psi dihedrals for residues 6 to 9:

```
multidihedral phi psi resrange 6-9 out PhiPsi_6-9.dat
```

Dihedrals other than those defined in **<dihedral types>** can be searched for using **dihtype**. For example to create a custom dihedral type called chi1 using atoms N, CA, CB, and CG (all in the same residue), then search for and calculate the dihedral in all residues:

```
multidihedral dihtype chi1:N:CA:CB:CG out custom.dat
```

28.9.47. multivector

```
multivector [<name>] [resrange <range>] name1 <name1> name2 <name2> [out <filename>]
            [ired]
```

[<name>] `Data set name.`

[resrange <range>] `Range of residues to look for vectors in.`

name1 <name1> `Name of first atom in each residue.`

name2 <name2> `Name of second atom in each residue.`

[out <filename>] `File to write results to.`

Search for and calculate atomic vectors between atoms named <name1> and <name2> in residues specified by the given <range>; each one is equivalent to the command '`vector <name1> <name2>`'. For example, to calculate all vectors between atoms named 'N' and atoms named 'H' in residues 5-20, storing the results in data sets named NH and writing to NH.dat:

```
multivector NH name1 N name2 H ired out NH.dat resrange 5-20
```

28.9.48. nastruct

```
nastruct [<dataset name>] [resrange <range>] [naout <suffix>]
      [noheader] [resmap <ResName>:{A,C,G,T,U} ...] [calcnohb]
         [baseref <file>] ...
      [hbcut <hbcut>] [origincut <origincut>] [altona | cremer]
      [zcut <zcut>] [zanglecut <zanglecut>] [groovecalc {simple | 3dna}]
      [{ reference | ref <name> | refindex <#> | allframes}]
```

[<dataset name>] `Output data set name.`

[resrange <range>] `Residue range to search for nucleic acids in (default all).`

[naout <suffix>] `File name suffix for output files; BP.<suffix> for base pair parameters, BPstep.<suffix> for base pair step parameters, and Helix.<suffix> for base pair step helical parameters.`

[noheader] `Do not print header to naout file.`

[resmap <ResName>:{A,C,G,T,U}] `Attempt to treat residues named <ResName> as if it were A, C, G, T, or U; useful for residues with modifications or non-standard residue names.  This will only work if enough reference atoms are present in <ResName>.`

[calcnohb] `Calculate parameters between bases in base pairs even if no hydrogen bonds present between them.`

[baseref <file>] `Specify a custom nucleic acid base reference.  One file per custom residue; multiple 'baseref' keywords may be present.  See below for details.`

[hbcut <hbcut>] `Distance cutoff (in Angstroms) for determining hydrogen bonds between bases (default 3.5).`

[origincut <origincut>] `Distance cutoff (in Angstroms) between base pair axis origins for determining which bases are eligible for base-pairing (default 2.5).`

[altona] `Use method of Altona & Sundaralingam to calculate sugar pucker (default, see` *pucker* `command).`

[cremer] `Use method of Cremer and Pople to calculate sugar pucker (see` *pucker* `command).`

[zcut] `Distance cutoff (in Angstroms) between base reference axes along the Z axis (i.e.  stagger) for determining base pairing (default 2).`

[zanglecut] `Angle cutoff (in degrees) between base reference Z axes for determining base pairing (default 65).`

[groovecalc] Groove width calculation method:

- **simple** Use P-P distance for major groove, O4-O4 distance for minor groove. Output to 'BP.<suffix>'.
- **3dna** Use groove width calculation of El Hassan and Calladine[568]. Output to 'BPstep.<suffix>'.

[reference | refindex <#> | ref <name>] Reference structure to use to determine base pairing; if not specified use first frame.

[allframes] If specified determine base pairing each frame.

DataSets Created:

<name>[pucker]:X Base X (residue number) sugar pucker.

Base pairs:

<name>[shear]:X Base pair X (starting from 1) shear.

<name>[stretch]:X Base pair stretch.

<name>[stagger]:X Base pair stagger.

<name>[buckle]:X Base pair buckle.

<name>[prop]:X Base pair propeller.

<name>[open]:X Base pair opening.

<name>[hb]:X Number of WC hydrogen bonds between bases in base pair.

<name>[bp]:X Contain 1 if bases are base paired, 0 otherwise.

<name>[major]:X (If groovecalc simple) Major groove width calculated between P atoms of each base.

<name>[minor]:X (If groovecalc simple) Minor groove width calculated between O4 atoms of each base.

Base pair steps:

<name>[shift]:X Base pair step X (starting from 1) shift.

<name>[slide]:X Base pair step slide.

<name>[rise]:X Base pair step rise.

<name>[title]:X Base pair step tilt.

<name>[roll]:X Base pair step roll.

<name>[twist]:X Base pair step twist.

<name>[zp]:X Base pair step Zp value.

<name>[major]:X (If groovecalc 3dna) Major groove width, El Hassan and Calladine.

<name>[minor]:X (If groovecalc 3dna) Minor groove width, El Hassan and Calladine.

Helical steps:

<name>[xdisp]:X Helical step X (starting from 1) X displacement.

<name>[ydisp]:X Helical Y displacement.

<name>[hrise]:X Helical rise.

<name>[incl]:X Helical inclination.

<name>[tip]:X Helical tip.

<name>[htwist]:X Helical twist.

Note that data sets are not created until base pairing is determined.

Calculate basic nucleic acid (NA) structure parameters for all residues in the range specified by **resrange** (or all NA residues if no range specified). Residue names are recognized with the following priority: standard Amber residue names DA, DG, DC, DT, RA, RG, RC, and RU; 3 letter residue names ADE, GUA, CYT, THY, and URA; and finally 1 letter residue names A, G, C, T, and U. Non-standard/modified NA bases can be recognized by using the **resmap** keyword. For example, to make *cpptraj* recognize all 8-oxoguanine residues named '8OG' as a guanine-based residue:

```
nastruct naout nastruct.dat resrange 274-305 resmap 8OG:G
```

The **resmap** keyword can be specified multiple times, but only one mapping per unique residue name is allowed. Note that **resmap** may fail if the residue is missing heavy atoms normally present in the specified base type.

Base pairs are determined either once from the first frame or from a reference structure, or can be determined each frame if **allframes** is specified. Base pairing is determined first by base reference axis origin distance, then by stagger, then by angle between base Z axes, then finally by hydrogen bonding (at least one hydrogen bond must be present). Base pair parameters will only be written for determined base pairs. Both Watson-Crick and other types of base pairing can be detected. Note that although all possible hydrogen bonds are searched for, only WC hydrogen bonds are reported in the BP.<suffix> file.

The procedure used to calculate NA structural parameters is the same as 3DNA[569], with algorithms adapted from Babcok et al.[570] and reference frame coordinates from Olson et al.[571]. Given the same base pairs are determined, *cpptraj* nastruct gives the exact same numbers as 3DNA.

Calculated NA structure parameters are written to three separate files, the suffix of which is specified by **naout**. Base pair parameters (shear, stretch, stagger, buckle, propeller twist, opening, # WC hydrogen bonds, base pairing, and simple groove widths) are written to BP.<suffix>, along with the number of WC hydrogen bonds detected. Base pair step parameters (shift, slide, rise, tilt, roll, twist, Zp, and El Hassan and Calladine groove widths) are written to BPstep.<suffix>, and helical parameters (X-displacement, Y-displacement, rise, inclination, tip, and twist) are written to Helix.<suffix>. If **noheader** is specified a header will not be written to the output files. Note that although base puckering is calculated, it is not written to an output file by default. You can output pucker to a file via the create or write/writedata commands after the data has been generated, e.g.:

```
nastruct NA naout nastruct.dat resrange 1-3,28-30
run
writedata NApucker.dat NA[pucker]
```

Custom Nucleic Acid Base References

Users can now specify **baseref <file>** to load a custom nucleic acid base reference. The base reference files are white-space delimited, begin with the line NASTRUCT REFERENCE, and have the following format:

```
NASTRUCT REFERENCE
<base character> <res name 0> [<res name 1> ...]
<atom name> <X> <Y> <Z> <HB type> <RMS fit>
...
```

There is a line for each reference atom. Lines beginning with '#' are ignored as comments.

<base character> Used to identify the underlying base type: A G C T or U. If none of these, it will be considered an unknown residue (which just means WC hydrogen bonding will not be identified).

<res name X> Specifies what residue names this reference corresponds to. There must be at least one residue name. There can be any number of these specified.

<atom name> A reference atom name.

<X> <Y> <Z> The X Y and Z coordinates of the reference atom.

<HB type> Denotes if and how the atom participates in hydrogen bonding. Can be 'd'onor, 'a'cceptor, or 'n'one (or the numbers 1, 2, 0 respectively). Only the first character of the word actually matters.

<RMS fit> Denotes whether the atom is involved in RMS-fitting.

Here is an example for GUA:

```
NASTRUCT REFERENCE
G G G5 G3
# Modified into format readable by cpptraj nastruct
C1' -2.477   5.399   0.000 0      0
N9  -1.289   4.551   0.000 0      1
C8   0.023   4.962   0.000 0      1
N7   0.870   3.969   0.000 accept 1
C5   0.071   2.833   0.000 0      1
C6   0.424   1.460   0.000 0      1
O6   1.554   0.955   0.000 accept 0
N1  -0.700   0.641   0.000 donor  1
C2  -1.999   1.087   0.000 0      1
N2  -2.949   0.139  -0.001 donor  0
N3  -2.342   2.364   0.001 accept 1
C4  -1.265   3.177   0.000 0      1
```

28.9.49. nativecontacts

```
nativecontacts [<mask1> [<mask2>]] [writecontacts <outfile>] [resout <resfile>]
               [noimage] [distance <cut>] [out <filename>] [includesolvent]
               [ first | reference | ref <name> | refindex <#> ]
               [resoffset <n>] [contactpdb <file>] [pdbcut <cut>] [mindist] [maxdist]
               [name <dsname>] [byresidue] [map [mapout <mapfile>]] [series [seriesout <file>]]
               [savenonnative [seriesnnout <file>] [nncontactpdb <file>]]
               [resseries { present | sum } [resseriesout <file>]]
```

<mask1> `First mask to calculate contacts for.`

[<mask2>] `(Optional) Second mask to calculate contacts for.`

[writecontacts <outfile>] `Write information on native contacts to <outfile> (STDOUT if not specified).`

[resout <resfile>] `File to write contact residue pairs to.`

[noimage] `Do not image distances.`

[distance <cut>] `Distance cutoff for determining native contacts in Angstroms (default 7.0 Ang).`

[out <filename>] `File to write number of native contacts and non-native contacts.`

[includesolvent] `By default solvent molecules are ignored; this will explicitly include solvent molecules.`

[first | reference | ref <name> | refindex <#>] `Reference structure to use for determining native contacts.`

[resoffset <n>] `(byresidue only) Ignore contacts between residues spaced less than <n> residues apart in sequence.`

[contactpdb <file>] `Write PDB with B-factor column containing relative contact strength for native contacts (strongest is 100.0).`

[pdbcut <cut>] `If writing contactpdb, only write contacts with relative contact strength greater than <cut>.`

[mindist] If specified, determine the minimum distance between any atoms in the mask(s).

[maxdist] If specified, determine the maximum distance between any atoms in the mask(s).

[name <dsname>] Data set name.

[byresidue] Write out the contact map by residue instead of by atom.

[map] Calculate matrices of native contacts ([nativemap]) and non-native contacts ([nonnatmap]). These matrices are normalized by the total number of frames, so that a value of 1.0 means "contact always present". If byresidue specified, the values for each individual atom pair are summed over the residues they belong to (this means for byresidue values greater than 1.0 are possible).

[mapout <mapfile>] Write native/non-native matrices to 'native.<mapfile>' and 'nonnative.<mapfile>' respectively.

[series] Calculate native contact time series data, 1 for contact present and 0 otherwise.

[seriesout <file>] Write native contact time series data to file.

[savenonnative] Save non-native contacts; series must also be specified.

  [seriesnnout <file>] Write non-native contact time series data to file.

  [nncontactpdb <file>] Write PDB with B-factor column containing relative contact strength for non-native contacts (strongest is 100.0).

[resseries {present | sum} Create contacts time series by residue; series must also be specified.

  present Record a 1 if *any* contact is present and 0 if no contact is present for the residue pair.

  sum The sum of all individual contacts is recorded for the residue pair.

  [resseriesout <file>] Write residue time series data to <file>.

Data Sets Created:

<dsname>[native] Number of native contacts.

<dsname>[nonnative] Number of non-native contacts.

<dsname>[mindist] (mindist only) Minimum observed distance each frame.

<dsname>[maxdist] (maxdist only) Maximum observed distance each frame.

<dsname>[nativemap] (map only) Native contacts matrix (2D).

<dsname>[nonnatmap] Non-native contacts matrix (2D).

<dsname>[NC] Native contacts time series.

<dsname>[NN] Non-native contacts time series.

<dsname>[NCRES] Residue native contacts time series.

<dsname>[NNRES] Residue non-native contacts time series.

Define and track "native" contacts as determined by a simple distance cut-off, i.e. any atoms which are closer than **<cut>** in the specified reference frame (the first frame if no reference specified) are considered a native contact. If one mask is provided, contacts are looked for within **<mask1>**; if two masks are provided, only contacts between atoms in **<mask1>** and atoms in **<mask2>** are looked for (useful for determining intermolecular contacts). By default only native contacts are tracked. This can be changed by specifying the **savenonnative** keyword. The time series for contacts can be saved using the **series** keyword; these can be further consolidated by residue using the **resseries** keyword. When using <resseries> the data set index is calculated as (r2 * nres) + r1 so that indices can be matched between native/non-native contact pairs. Non-native residue contact legends have an nn_ prefix.

Native contacts that are found are written to the file specified by writecontacts (or STDOUT) with format:

```
# Contact Nframes Frac. Avg Stdev
```

Where `Contact` takes the form ':<residue1 num>@<atom name>_:<residue2 num>@<atom name>, `Nframes` is the number of frames the contact is present, `Frac.` is the total fraction of frames the contact is present, `Avg` is the average distance of the contact when present, and `Stdev` is the standard deviation of the contact distance when present. If **resout** is specified the total fraction of contacts is printed for all residue pairs having native contacts with format:

```
#Res1 #Res2 TotalFrac Contacts
```

Where `#Res1` is the first residue number, `#Res2` is the second residue number, `TotalFrac` is the total fraction of contacts for the residue pair, and `Contacts` is the total number of native contacts involved with the residue pair. Since `TotalFrac` is calculated for each pair as the sum of each contact involving that pair divided by the total number of frames, it is possible to have `TotalFrac` values greater than 1 if the residue pair includes more than 1 native contact.

During trajectory processing, non-native contacts (i.e. any pair satisfying the distance cut-off which is not already a native contact) are also searched for. The time series for native contacts can be saved as well, with 1 for contact present and 0 otherwise (similar to the ***hbond*** command). This data can be subsequently analyzed using e.g. 28.10.17 on page 665.

Contact maps (matrices) are generated for native and non-native contacts. If **byresidue** is specified, contact maps are summed over residues, and contacts between residues spaced **<resoffset>** residues apart in sequence are ignored.

If **contactpdb** is specified a PDB is generated containing relative contact strengths in the B-factor column. The relative contact strength is normalized so that a value of 100 means that atom participated in the most contacts with other atoms.

Example command looking for contacts between residues 210 to 260 and residue named NDP, using reference structure 'FtuFabI.WT.pdb' to define native contacts:

```
parm FtuFabI.parm7
trajin FtuFabI.nc
reference FtuFabI.WT.pdb
nativecontacts name NC1 :210-260&!@H= :NDP&!@H= \
    byresidue out nc.all.res.dat mindist maxdist \
    distance 3.0 reference map mapout resmap.gnu \
    contactpdb Loop-NDP.pdb \
    series seriesout native.dat
```

28.9.50. outtraj

```
outtraj <filename> [ trajout args ]
        [maxmin <dataset> min <min> max <max>] ...
```

<filename> `Output trajectory file name.`

[trajout args] `Output trajectory arguments (see 28.8.5 on page 584).`

[maxmin <dataset> min <min> max <max>] `Only write frames to <filename> if values in <dataset> for those frames are between <min> and <max>. Can be specified for one or more data sets.`

The outtraj command is similar in function to ***trajout***, and takes all of the same arguments. However, instead of writing a trajectory frame after all actions are complete outtraj writes the trajectory frame at its position in the Action queue. For example, given the input:

```
trajin mdcrd.crd
trajout output.crd
outtraj BeforeRmsd.crd
rms R1 first :1-20@CA out rmsd.dat
outtraj AfterRmsd.crd
```

three trajectories will be written: output.crd, BeforeRmsd.crd, and AfterRmsd.crd. The output.crd and AfterRmsd.crd trajectories will be identical, but the BeforeRmsd.crd trajectory will contain the coordinates of mdcrd.crd before they are RMS-fit.

The maxmin keyword can be used to restrict output using one more more data sets. For example, to only write frames for which the RMSD value is between 0.7 and 0.8:

```
trajin tz2.truncoct.nc
rms R1 first :2-11
outtraj maxmin.crd maxmin R1 min 0.7 max 0.8
```

28.9.51. pairdist

```
pairdist out <filename> mask <mask> [delta <resolution>]
```

Calculate pair distribution function. In the following, defaults are given in parentheses. The **out** keyword specifies output file for histogram: distance, P(r), s(P(r)). The **mask** option specifies atoms for which distances should be computed. The **delta** option specifies resolution. (0.1 Å)

28.9.52. pairwise

```
pairwise [<name>] [<mask>] [out <filename>] [cuteelec <ecut>] [cutevdw <vcut>]
          [ reference | ref <name> | refindex <#> ] [cutout <cut mol2 prefix>]
       [vmapout <vdw map>] [emapout <elec map>] [avgout <avg file>]
       [eout <eout file>] [pdbout <pdb file>]
```

[<name>] `Data set name; van der Waals energy will get aspect [EVDW] and electrostatic energy will get aspect [EELEC].`

[<mask>] `Atoms to calculate energy for.`

[out <filename>] `File to write total EELEC and EVDW to.`

[eout <eout file>] `File to write individual EELEC and EVDW interactions to.`

[reference | ref <name> | refindex <#>] `Specify a reference to compare frames to (i.e. calculate Eref - Eframe).`

[cuteelec <cut>] `Only report interaction EELEC (or delta EELEC) if absolute value is greater than <ecut> (default 1.0 kcal/mol).`

[cutevdw <cutv>] `Only report interaction EVDW (or delta EVDW) if absolute value is greater than <vcut> (default 1.0 kcal/mol).`

[cutout <cut mol2 prefix>] `Write out mol2 containing only atom pairs which satisfy <ecut> and <vcut>.`

[vmapout <vdw map>] `Write out interaction EVDW (or delta EVDW) matrix to file <vdw map>.`

[emapout <elec map>] `Write out interaction EELEC (or delta EELEC) matrix to file <elec map>.`

[avgout <avg file>] `Print average interaction EVDW|EELEC (or average delta EVDW|EELC) to <avg file>.`

[pdbout <pdb file>] `Write PDB with EVDW|EELEC in occupancy|B-factor columns to <pdb file>.`

`Data Sets Created:`

<name>[EELEC] `Electrostatic energy in (kcal/mol).`

<name>[EVDW] `van der Waals energy in (kcal/mol).`

<name>[VMAP] `van der Waals energy matrix.`

<name>[EMAP] `Electrostatic energy matrix.`

This action has two related functions: 1) Calculate pairwise (i.e. non-bonded) energy (in kcal/mol) for atoms in **<mask>**, or 2) Compare pairwise energy of frames to a reference frame. This calculation does use an exclusion list but is not periodic.

When comparing to a reference frame, the **eout** file will contain the differences for each individual interaction (i.e. Eref - Eframe), otherwise the **eout** file will contain the absolute value of each individual interaction. The **cuteelc** and **cutevdw** keywords can be used to restrict printing of individual interactions to those for which the absolute value is above a cutoff. The VMAP and EMAP matrix elements will contain these values as well (differences for reference, absolute value otherwise) averaged over all frames. The **avgout** file will contain only these values averaged over all frames that satisfy the cutoffs.

The **cutout** keyword can be used to write out MOL2 files each frame named '<cut mol2 prefix>.evdw.mol2.X' and '<cut mol2 prefix>.eelec.mol2.X' (where X is the frame number) containing only atoms with energies that satisfy the cutoffs. Similarly, the **pdbout** keyword can be used to write out a PDB file (with 1 MODEL per frame). The occupancy and B-factor columns will contain the total van der Waals and electrostatic energy for each atom if cutoffs are satisfied, or 0.0 otherwise.

28.9.53. principal

```
principal [<mask>] [dorotation] [out <filename>] [name <dsname>]
```

[<mask>] `Mask of atoms used to determine principal axes (default all).`

[dorotation] `Align coordinates along principal axes.`

[out <filename>] `Write resulting eigenvalues/eigenvectors to <filename>.`

[name <dsname>] `Data set name (3x3 matrices).`

`Data Sets Created (name keyword only):`

<dsname>[evec] `Eigenvectors (3x3 matrix, row-major).`

<dsname>[eval] `Eigenvalues (vector).`

Determine principal axes of each frame determined by diagonalization of the inertial matrix from the coordinates of the specified atoms. At least one of **dorotation**, **out**, or **name** must be specified. The resulting eigenvectors are sorted from largest eigenvalue to smallest, and the corresponding axes labelled using the *cpptraj* convention of X > Y > Z (similar to '***vector*** **principal**'). If out is specified the eigenvectors and eigenvalues will be written for each frame N with format:

```
<N> EIGENVALUES: <EX> <EY> <EZ>
<N> EIGENVECTOR 0: <Xx> <Xy> <Xz>
<N> EIGENVECTOR 1: <Yx> <Yy> <Yz>
<N> EIGENVECTOR 2: <Zx> <Zy> <Zz>
```

NOTE: The eigenvector 3x3 matrix data set could subsequenetly be used e.g. with the ***rotate*** action.

Example: Align system (residues 1-76) along principle axes:

```
parm myparm.parm7
trajin protein.nc
principal :1-76 dorotation out principal.dat
```

28.9.54. projection

```
projection [<name>] evecs <dataset name> [out <outfile>] [beg <beg>] [end <end>]
           [<mask>] [dihedrals <dataset arg>]
           [start <start>] [stop <stop>] [offset <offset>]
```

[<name>] Output data set name.

evecs <dataset name> Data set containing eigenvectors (modes).

[out <outfile>] Write projections to <outfile>.

[beg <beg>] First eigenvector/mode to use (default 1).

[end <end>] Final eigenvector/mode to use (default 2).

[<mask>] (Not dihedral covariance) Mask of atoms to use in projection; MUST CORRESPOND TO HOW EIGENVECTORS WERE GENERATED.

[dihedrals <dataset arg>] (Dihedral covariance only) Dihedral data sets to use in projection; MUST CORRESPOND TO HOW EIGENVECTORS WERE GENERATED.

[start <start>] Frame to start calculating projection.

[stop <stop>] Frame to stop calculating projection.

[offset <offset>] Frames to skip between projection calculations.

Data Sets Created:

DataSet indices correspond to mode #.

<name> (All execpt IDEA) Projection data set.

<name>[X] X component of mode (IDEA modes only).

<name>[Y] Y component of mode (IDEA modes only).

<name>[Z] Z component of mode (IDEA modes only).

<name>[R] Magnitude of mode (IDEA modes only).

Projects snapshots onto eigenvectors obtained by diagonalizing covariance or mass-weighted covariance matrices. Eigenvectors are taken from previously generated (e.g. with ***diagmatrix***) or previously read-in (e.g. with ***readdata***) eigenvectors with name <dataset name>. The user has to make sure that the atoms selected by **<mask>** agree with the ones used to calculate the modes (i.e., if mask = '@CA' was used in the "**matrix**" command, mask = '@CA' needs to be set here as well). See 28.11 on page 682 for examples using the ***projection*** command.

28.9.55. pucker

```
pucker [<name>] <mask1> <mask2> <mask3> <mask4> <mask5> [<mask6>] [geom]
   [out <filename>] [altona | cremer] [amplitude] [theta]
    [range360] [offset <offset>]
```

<name> Output data set name.

<maskX> Five (optionally six) atom masks selecting atom(s) to calculate pucker for.

[geom] Use geometric center of atoms in <maskX> (default is center of mass).

[out <filename>] Output file name.

[altona] Use method of Altona & Sundaralingam (5 masks only).

[cremer] Use method of Cremer and Pople (5 or 6 masks).

[amplitude] Also calculate amplitude.

[theta] (6 masks only) Also calculate theta.

[range360] Wrap pucker values from 0.0 to 360.0 (default is -180.0 to 180.0).

[offset <offset>] Add <offset> to pucker values.

Data Sets Created:

<name> Pucker in degrees.

<name>[Amp] `Amplitude (if amplitude was specified).`

<name>[Theta] `Theta (if theta and 6 masks were specified).`

Calculate the pucker (in degrees) for atoms in <mask1>, <mask2>, <mask3>, <mask4>, <mask5> using the method of Altona & Sundarlingam[572, 573] (default, or if **altona** specified), or the method of Cremer & Pople[574] if **cremer** is specified. If <mask6> is specified calculate the pucker (and optionally theta if **theta** specified) according to the method of Cremer & Pople. If the **amplitude** keyword is given, amplitudes will be calculated in addition to pucker. The results from ***pucker*** can be further analyzed with the ***statistics*** analysis.

By default, pucker values are wrapped to range from -180 to 180 degrees. If the **range360** keyword is specified values will be wrapped to range from 0 to 360 degrees. Note that the Cremer & Pople convention is offset from Altona & Sundarlingam convention (with nucleic acids) by +90.0 degrees; the **offset** keyword will add an offset to the final value and so can be used to convert between the two. For example, to convert from Cremer to Altona specify "**offset** 90".

To calculate nucleic acid pucker specify C1' first, followed by C2', C3', C4' and O4'. For example, to calculate the sugar pucker for nucleic acid residues 1 and 2 using the method of Altona & Sundarlingam, with final pseudorotation values ranging from 0 to 360:

```
pucker p1 :1@C1' :1@C2' :1@C3' :1@C4' :1@O4' range360 out pucker.dat
pucker p2 :2@C1' :2@C2' :2@C3' :2@C4' :2@O4' range360 out pucker.dat
```

28.9.56. radgyr | rog

```
radgyr [name>] [<mask>] [out <filename>] [mass] [nomax] [tensor]
```

[<name>] `Data set name.`

[<mask>] `Atoms to calculate radius of gyration for; default all atoms.`

[out <filename>] `Write data to <filename>.`

[mass] `Mass-weight radius of gyration.`

[nomax] `Do not calculate maximum radius of gyration.`

[tensor] `Calculate radius of gyration tensor, output format 'XX YY ZZ XY XZ YZ'.`

`Data Sets Created:`

<name> `Radius of gyration in Ang.`

<name>[Max] `Max radius of gyration in Ang.`

<name>[Tensor] `Radius of gyration tensor; format 'XX YY ZZ XY XZ YZ'.`

Calculate the radius of gyration of specified atoms. For example, to calculate only the mass-weighted radius of gyration (not the maximum) of the non-hydrogen atoms of residues 4 to 10 and print the results to "RoG.dat":

```
radgyr :4-10&!(@H=) out RoG.dat mass nomax
```

28.9.57. radial | rdf

```
radial <outfilename> <spacing> <maximum> <solvent mask1>
       [<solute mask2>] [noimage] [density <density> | volume]
       [center1 | center2 | nointramol] [<name>]
       [intrdf <file>] [rawrdf <file>]
```

<outfilename> `File to write RDF to, required.`

<spacing> `Bin spacing, required.`

<maximum> `Max bin value, required.`

<solvent mask1> `Atoms to calculate RDF for, required.`

[<solute mask2>] `(Optional) If specified calculate RDF of all atoms in <solvent mask1> to each atom in <solute mask2>.`

[noimage] `Do not image distances.`

[density <density>] `Use density value of <density> for normalization (default 0.033456 molecules Å`$^{-3}$`).`

[volume] `Determine density for normalization from average volume of input frames.`

[center1] `Calculate RDF from geometric center of atoms in <solvent mask1> to all atoms in <solute mask2>.`

[center2] `Calculate RDF from geometric center of atoms in <solute mask2> to all atoms in <solvent mask1>.`

[nointramol] `Ignore intra-molecular distances.`

[<name>] `Name radial dataset.`

[intrdf <file>] `Calculate integral of RDF bin values (averaged over # of frames but otherwise not normalized) and write to <file> (can be same as <output_filename>).`

[rawrdf <file>] `Write raw (non-normalized) RDF values to <file>.`

`DataSet Aspects:`

[int] `(intrdf only) Integral of RDF bin values.`

[raw] `(rawrdf only) Raw (non-normalized) RDF values.`

Calculate the radial distribution function (RDF, aka pair correlation function) of atoms in **<solvent mask1>** (note that this mask does not need to be solvent, but this nomenclature is used for clarity). If an optional second mask (**<solute mask2>**) is given, calculate the RDF of ALL atoms in **<solvent mask1>** to EACH atom in **<solute mask2>**. If desired, the geometric center of atoms in **<solvent mask1>** or **<solute mask2>** can be used by specifying the **center1** or **center2** keywords respectively, or alternatively intra-molecular distances can be ignored by specifying the **nointramol** keyword.

The RDF is calculated from the histogram of the number of particles found as a function of distance R, normalized by the expected number of particles at that distance. The normalization is calculated from:

$$Density * \left(\left[\tfrac{4\pi}{3} (R+dR)^3 \right] - \left[\tfrac{4\pi}{3} dR^3 \right] \right)$$

where dR is equal to the bin spacing. Some care is required by the user in order to normalize the RDF correctly. The default density value is 0.033456 molecules Å^{-3}, which corresponds to a density of water approximately equal to 1.0 g mL^{-1}. To convert a standard density in g mL^{-1}, multiply the density by $\frac{0.6022}{M_r}$, where M_r is the mass of the molecule in atomic mass units. Alternatively, if the **volume** keyword is specified the density is determined from the average volume of the system over all Frames.

Note that correct normalization of the RDF depends on the number of atoms in each mask; if multiple topology files are being processed that result in changes in the number of atoms in each mask, the normalization will be off.

28.9.58. randomizeions

```
randomizeions <mask> [around <mask> by <distance>] [overlap <value>]
              [noimage] [seed <value>]
```

This can be used to randomly swap the positions of solvent and single atom ions. The "`overlap`" specifies the minimum distance between ions, and the "`around`" keyword can be used to specify a solute (or set of atoms) around which the ions can get no closer than the distance specified. The optional keywords "`noimage`" disable imaging and "`seed`" update the random number seed. An example usage is

```
randomizeions @NA around :1-20 by 5.0 overlap 3.0
```

The above will swap Na$^+$ ions with water getting no closer than 5.0 Å from residues 1 – 20 and no closer than 3.0 Å from any other Na$^+$ ion.

28.9.59. replicatecell

```
replicatecell [out <traj filename>] [parmout <parm filename>] [name <dsname>]
              { all | dir <XYZ> [dir <XYZ> ...] } [<mask>]
```

out <traj filename> Write replicated cell to output trajectory file.

parmout <parm filename> Write replicated cell topology to topology file. This file will not be viable to use for simulations.

name <dsname> If specified save replicated cell to COORDS data set.

all Replicate cell once in all possible directions.

dir <XYZ> Repicate cell once in specified directions. <XYZ> should consist of 3 numbers with no spaces in between them and are restricted to values of -1, 1, and 0. May be specified more than once.

<mask> Mask of atoms to replicate.

Create a trajectory where the unit cell is replicated in 1 or more directions (up to 27). The resulting coordinates and topology can be written to a trajectory/topology file. They can also be saved as a COORDS data set for subsequent processing. Currently replication is only allowed 1 axis length in either direction. The **all** keyword will replicate the cell once in all directions. The **dir** keyword can be used to restrict replication to specific directions, e.g. '`dir 10-1`' would replicate the cell once in the +X, -Z directions.

For example, to replicate a cell in all directions, writing out to NetCDF trajectory cell.nc:

```
parm ../tz2.truncoct.parm7
trajin ../tz2.truncoct.nc
replicatecell out cell.nc parmout cell.parm7 all
```

28.9.60. rms | rmsd

```
rmsd [<name>] <mask> [<refmask>] [out filename] [nofit | norotate | nomod]
  [mass] [savematrices]
  [ first | reference | ref <name> | refindex <#> | previous |
    reftraj <name> [parm <name> | parmindex <#>] ]
  [perres perresout <filename> [perresavg <avgfile>]
   [range <resRange>] [refrange <refRange>]
   [perresmask <additional mask>] [perrescenter] [perresinvert]
```

[<name>] Output data set name.

[<mask>] Mask of atoms to calculate RMSD for; if not specified, calculate for all atoms.

[<refmask>] Reference mask; if not specified, use <mask>.

[out <filename>] Output data file name.

[nofit] Do not perform best-fit RMSD.

[norotate] If calculating best-fit RMSD, translate but do not rotate coordinates.

[nomod] If calculating best-fit RMSD, do not modify coordinates.

[mass] Mass-weight the RMSD calculation.

[savematrices] If specified save rotation matrices to data set with aspect [RM].

Reference keywords:

first Use the first trajectory frame processed as reference.

reference Use the first previously read in reference structure (refindex 0).

ref <name> `Use previously read in reference structure specified by filename/tag.`

refindex <#> `Use previously read in reference structure specified by <#> (based on order read in).`

previous `Use frame prior to current frame as reference.`

reftraj <name> `Use frames from COORDS set <name> or read in from trajectory file <name> as references. Each frame from <name> is used in turn, so that frame 1 is compared to frame 1 from <name>, frame 2 is compared to frame 2 from <name> and so on. If <trajname> runs out of frames before processing is complete, the last frame of <trajname> continues to be used as the reference.`

  parm <parmname> | parmindex <#> `If reftraj specifies a trajectory file, associate it with specified topology; if not specified the first topology is used.`

`Per-residue RMSD keywords:`

perres `Activate per-residue no-fit RMSD calculation.`

perresout <perresfile> `Write per-residue RMSD to <perresfile>.`

perresavg <avgfile> `Write average per-residue RMSDs to <avgfile>.`

range <res range> `Calculate per-residue RMSDs for residues in <res range> (default all solute residues).`

refrange <ref range> `Calculate per-residue RMSDs to reference residues in <ref range> (use <res range> if not specified).`

perresmask <additional mask> `By default residues are selected using the mask ':X' where X is residue number; this appends <additional mask> to the mask expression.`

perrescenter `Translate residues to a common center of mass prior to calculating RMSD.`

perresinvert `Make X-axis residue number instead of frame number.`

`Data Sets Created:`

<name> `RMSD of atoms in mask to reference.`

<name>[RM] `(savematrices only) Rotation matrices of target to reference.`

<name>[res] `(perres only) Per-residue RMSDs; index is residue number.`

<name>[Avg] `(perres only) Average per-residue RMSD for each residue.`

<name>[Stdev] `(perres only) Standard deviation of RMSD for each residue.`

*Note that **perres** data sets are not generated until **run** is called.*

Calculate the coordinate RMSD of input frames to a reference frame (or reference trajectory). Both <mask> and <refmask> must specify the same number of atoms, otherwise an error will occur.

For example, say you have a trajectory and you want to calculate RMSD to two separate reference structures. To calculate the best-fit RMSD of the C, CA, and N atoms of residues 1 to 20 in each frame to the C, CA, and N atoms of residues 3 to 23 in StructX.crd, and then calculate the no-fit RMSD of residue 7 to residue 7 in another structure named Struct-begin.rst7, writing both results to Grace-format file “rmsd1.agr”:

```
reference StructX.crd [structX]
reference md_begin.rst7 [struct0]
rmsd BB :1-20@C,CA,N ref [structX] :3-23@C,CA,N out rmsd1.agr
rmsd Res7 :7 ref [struct0] out rmsd1.agr nofit
```

Per-residue RMSD calculation

If the **perres** keyword is specified, after the initial RMSD calculation the no-fit RMSD of specified residues is also calculated. So for example:

```
rmsd :10-260 reference perres perresout PRMS.dat range 190-211 perresmask &!(@H=)
```

will first perform a best-fit RMSD calculation to the first specified reference structure using residues 10 to 260, then calculate the no-fit RMSD of residues 190 to 211 (excluding any hydrogen atoms), writing the results to PRMS.dat. Two additional recommendations for the 'perres' option: 1) try not including backbone atoms by using the 'perresmask' keyword, e.g. "perresmask &!@H,N,CA,HA,C,O", and 2) try using the 'perrescenter' keyword, which centers each residue prior to the 'nofit' calculation; this is useful for isolating changes in residue conformation.

28.9.61. rms2d | 2drms

Although the ***'rms2d'*** command can still be specified as an action, it is now considered an analysis. See 28.10.26 on page 672.

28.9.62. rmsavgcorr

Although the ***'rmsavgcorr'*** command can still be specified as an action, it is now considered an analysis. See 28.10.27 on page 673.

28.9.63. rmsf | atomicfluct

See 28.9.4 on page 591.

28.9.64. rotate

```
rotate [<mask>] { [x <xdeg>] [y <ydeg>] [z <zdeg>]  |
                  axis0 <mask0> axis1 <mask1> <deg> |
                  usedata <set name> [inverse] }
```

[<mask>] `Rotate atoms in <mask> (default all).`

[x <xdeg] `Degrees to rotate around the X axis.`

[y <xdeg] `Degrees to rotate around the Y axis.`

[z <xdeg] `Degrees to rotate around the Z axis.`

axis0 <mask0> `Mask defining the beginning of a user-defined axis.`

axis1 <mask1> `Mask defining the end of a user-defined axis.`

<deg> `Value in degrees to rotate around user defined axis.`

usedata <set name> `If specified, use 3x3 rotation matrices in specified data set to rotate coordinates.`

[inverse] `Perform inverse rotation from input rotation matrices.`

Rotate specified atoms around the X, Y, and/or Z axes by the specified amounts, around a user-defined axis (specified by <mask0> and <mask1>), or use a previously read in or generated data set of 3x3 matrices to perform rotations.

For example, to rotate the entire system 90 degrees around the X axis:

```
rotate x 90
```

To rotate residue 270 90 degrees around the axis defined between atoms C1, C2, C3, C4, C5, and C6 in residue 270 and atoms C7, C8, C9, C10, C11, and C12 in residue 270:

```
rotate :270 axis0 :270@C1,C2,C3,C4,C5,C6 axis1 :270@C7,C8,C9,C10,C11,C12 90.0
```

To rotate the system with rotation matrices read in from rmatrices.dat:

```
trajin tz2.norotate.crd
readdata rmatrices.dat name RM mat3x3
rotate usedata RM
```

28.9.65. rotdif

The ***'rotdif'*** command is now an analysis (see 28.10.28 on page 674), and requires that rotation matrices be generated via an ***rmsd*** action. For example:

```
reference avgstruct.pdb
trajin tz2.nc
rms R0 reference @CA,C,N,O savematrices
rotdif rmatrix R0[RM] rseed 1 nvecs 10 dt 0.002 tf 0.190 \
       itmax 500 tol 0.000001 d0 0.03 order 2 rvecout rvecs.dat \
       rmout matrices.dat deffout deffs.dat outfile rotdif.out
```

28.9.66. runavg | runningaverage

```
runavg [window <window_size>]
```

Note that for backwards compatibility with ptraj "runningaverage" is also accepted.

Replaces the current frame with a running average over a number of frames specified by **window** <window_size> (5 if not specified). This means that in order to build up the correct number of frames to calculate the average, the first <window_size> minus one frames will not be processed by subsequent actions. So for example given the input:

```
runavg window 3
rms first out rmsd.dat
```

the rms command will not take effect until frame 3 since that is the first time 3 frames are available for averaging (1, 2, and 3). The next frame processed would be an average of frames 2, 3, and 4, etc.

28.9.67. scale

```
scale x <sx> y <sy> z <sz> <mask>
```

Scale the X|Y|Z coordinates of atoms in <mask> by <sx>|<sy>|<sz>.

28.9.68. secstruct

```
secstruct [<name>] [out <filename>] [<mask>] [sumout <filename>]
          [assignout <filename>] [totalout <filename> [ptrajformat]
          [namen <N name>] [nameh <H name>] [nameca <CA name>]
          [namec <C name>] [nameo <O name>]
```

[<name>] `Output data set name.`

[out <filename>] `Output file name for secondary structure vs time.`

[<mask>] `Atom mask in which residues should be looked for.`

[sumout <sumfilename>] `Write average secondary structure values for each residue to <sumfilename>; if not specified <filename>.sum is used.`

[assignout <filename>] `Write overall secondary structure assignment (based on dominant secondary structure type for each residue) to file.`

[ptrajformat] `Write secondary structure as a string of characters for each frame, similar to ptraj output.`

[namen <N name>] `Backbone amide nitrogen atom name (default 'N').`

[nameh <H name>] `Backbone amide hydrogen atom name (default 'H').`

[nameca <CA name>] `Backbone alpha carbon atom name (default 'CA').`

[namec <C name>] `Backbone carbonyl carbon atom name (default 'C').`

[nameo <O name>] `Backbone carbonyl oxygen atom name (default 'O').`

`Data Sets Created:`

<name>[res] `Residue secondary structure per frame; index corresponds to residue number. If ptrajformat specified these will be characters, otherwise integers (see table below).`

<name>[avgss] `Average of each type of secondary structure; index corresponds to secondary structure type (see table below; no index for "None").`

<name>[None] `Total fraction of residues with no structure vs time.`

<name>[Para] `Total fraction of residues with parallel beta structure vs time.`

<name>[Anti] `Total fraction of residues with anti-parallel beta structure vs time.`

<name>[3-10] `Total fraction of 3-10 helical structure vs time.`

<name>[Alpha] `Total fraction of alpha helical structure vs time.`

<name>[Pi] `Total fraction of Pi helical structure vs time.`

<name>[Turn] `Total fraction of turn structure vs time.`

<name>[Bend] `Total fraction of bend structure vs time.`

Note that when not using ***ptrajformat****, data sets are not generated until* ***run*** *is called.*

Calculate secondary structural propensities for residues in **<mask>** (or all solute residues if no mask given) using the DSSP method of Kabsch and Sander[575], which assigns secondary structure types for residues based on backbone amide (N-H) and carbonyl (C=O) atom positions. By default *cpptraj* assumes these atoms are named “N”, “H”, “C”, and “O” respectively. If a different naming scheme is used (e.g. amide hydrogens are named “HN”) the backbone atom names can be customized with the **nameX** keywords (e.g. 'nameH HN'). Note that it is expected that some residues will not have all of these atoms (such as proline); in this case *cpptraj* will print an informational message but the calculation will proceed normally.

Results will be written to filename specified by **out** with format:

```
<#Frame>    <ResX SS> <ResX+1 SS> ... <ResN SS>
```

where <#Frame> is the frame number and <ResX SS> is an integer representing the calculated secondary structure type for residue X. If the keyword **ptrajformat** is specified, the output format will instead be:

```
<#Frame>    STRING
```

where STRING is a string of characters (one for each residue) where each character represents a different structural type (this format is similar to what *ptraj* outputs). The various secondary structure types and their corresponding integer/character are listed below:

Character	Integer	DSSP Char	SS type
0	0	’ ’	None
b	1	’E’	Parallel Beta-sheet
B	2	’B’	Anti-parallel Beta-sheet
G	3	’G’	3-10 helix
H	4	’H’	Alpha helix
I	5	’I’	Pi (3-14) helix
T	6	’T’	Turn
S	7	’S’	Bend

Average structural propensities over all frames for each residue will be written to the file specified by **sumout** (or “<*filename*>.*sum*” if **sumout** is not specified). The total structural propensity over all residues for each secondary structure type will be written to the file specified by **totalout**. If assignout is specified, the overall secondary structure assignment for each residue will be printed in two line chunks of 50 residues, with the first line containing the residue number the line starts with and one character residue names, and the second line containing secondary structure assignment using DSSP-style characters, like so:

```
1 KCNTATCATQ RLANFLVHSS NNFGAILSST NVGSNTRn
    SSS   TH HHHTTSBBBB TTTBBBB SS     S
```

The output of secstruct command is amenable to visualization with gnuplot. To generate a 2D map-style plot of secondary structure vs time, with each residue on the Y axis simply give the output file a “.gnu” extension. For example, to generate a 2D map of secondary structure vs time, with different colors representing different secondary structure types for residues 1-22:

```
secstruct :1-22 out dssp.gnu
```

The resulting file can be visualized with gnuplot:

```
gnuplot dssp.gnu
```

Similarly, the **sumout** file can be nicely visualized using xmgrace (use “.agr” extension).

```
secstruct :1-22 out dssp.gnu sumout dssp.agr
xmgrace dssp.agr
```

28.9.69. spam

```
spam <filename> [solv <solvname>] [reorder] [name <name>]
     [purewater] [cut <cut>] [info <infofile>] [summary <summary>]
     [site_size <size>] [sphere] [out <datafile>]
     [dgbulk <dgbulk>] [dhbulk <dhbulk>] [temperature <T>]
```

<filename> `File with the peak locations present (XYZ- format)`

<solvname> `Name of the solvent residues`

<cut> `Non-bonded cutoff for energy evaluation`

<dgbulk> `SPAM free energy of the bulk solvent in kcal/mol; default is -30.3 kcal/mol (SPC/E water).`

<dhbulk> `SPAM enthalpy of the bulk solvent in kcal/mol; default is -22.2 kcal/mol (SPC/E water).`

<T> `Temperature at which SPAM calculation was run.`

<infofile> `File with stats about which sites are occupied when.`

<size> `Size of the water site around each density peak.`

[sphere] `Treat each site like a sphere.`

[purewater] `The system is pure water---used to parametrize the bulk values.`

[reorder] `The solvent should be re-ordered so the same solvent molecule is always in the same site.`

<summary> `File with the summary of all SPAM results. If not specified, no SPAM energies will be calculated.`

<datafile> `Data file with all SPAM energies for each snapshot.`

Perform profiling of bound water molecules via SPAM analysis[576]. Briefly, this method identifies and estimates the free energy profiles of bound waters via calculation of the distribution of interaction energies between the water and it's environment from explicit solvent MD trajectories. The interaction energies are calculated using a force- and energy-shifted electrostatic term with a hard cutoff.

Prior to this command, the ***volmap*** command should be run with the **peakfile** keyword (see 28.9.81 on page 647) to generate the peaks file. If not using peaks from the ***volmap*** command, the peaks file should have one line per peak with format:

```
C <X> <Y> <Z> <Density>
```

Values of **dgbulk** and **dhbulk** for different water models can be calculated from pure water simulations with the **purewater** keyword.

28.9.70. setvelocity

```
setvelocity [<mask>] [tempi <temperature>] [ig <random seed>]
```

[<mask>] `Mask of atoms to assign velocities to.`

[tempi <temperature>] `Assign velocities at specified temperature (default 300.0 K).`

[ig <random seed>] `Random seed to use to generate velocity distribution.`

Set velocities in frame for atoms in <mask> using Maxwellian distribution based on given temperature.

28.9.71. stfcdiffusion

```
stfcdiffusion mask <mask> [out <file>] [time <time per frame>]
              [mask2 <mask>] [lower <distance>] [upper <distance>]
              [nwout <file>]) [avout <file>] [distances] [com]
              [x|y|z|xy|xz|yz|xyz]
```

mask `Atoms for which MSDs will be computed.`

out `Output file: time vs. MSD.`

time `Time step in the trajectory. (1.0 ps)`

mask2 `Compute MSDs only within the lower and upper limit of mask2. IMPORTANT: may be very slow!!!`

lower `Smaller distance from reference point(s). (0.01 Å)`

upper `Larger distance from reference point(s). (3.5 Å)`

nwout `Output file containing number of water molecules in the chosen region, see mask2. (off)`

avout `Output file containing average distances. (off)`

x|y|z|xy|xz|yz|xyz `Computation of the mean square displacement in the chosen dimension. (xyz)`

distances `Dump un-imaged distances. By default only averages are output. (off)`

com `Calculate MSD for centre of mass. (off)`

Calculate diffusion for selected atoms using code based on the 'diffusion' routine developed by Hannes Loeffler at STFC (http://www.stfc.ac.uk/CSE).

28.9.72. strip

```
strip <mask> [outprefix <name>] [nobox]
```

<mask> `Remove atoms specified by mask from the system.`

[outprefix <prefix>] `Write out stripped topology file with name '<prefix>.<Original Topology Name>'.`

[nobox] `Remove any box information from the stripped topology.`

Strip all atoms specified by **<mask>** from the frame and modify the topology to match for any subsequent Actions. The **outprefix** keyword can be used to write stripped topologies; stripped Amber topologies are fully-functional.

Note that stripping a system renumbers all atoms and residues, so for example after this command:

```
strip :1
```

residue 1 will be gone, and the former second residue will now be the first, and so on.

For example, to strip all residues named WAT from each topology/coordinate frame:

```
strip :WAT
```

The next example uses a distance-based mask to strip atoms in a single frame. Note that with the exception of the ***mask*** command, distance-based masks do not update on a per-frame basis. To strip all residues outside of 6.0 from any atom in residues 1 to 14 and write out the stripped topology and coordinates, both with no box information:

```
parm parm7
trajin frame_1000.rst.1
reference frame_1000.rst.1
strip !(:1-14<:6.0) outprefix f1.1 nobox
trajout f1.1.x restart nobox
```

28.9.73. surf

```
surf [<dataset name>] [<mask>] [out <filename>]
```

Calculate the surface area in Å^2 of atoms in <mask> (all solute atoms if no mask specified) using the LCPO algorithm of Weiser et al.[126]. In order for this to work, the topology needs to have bond information and atom type information.

Note that even if <mask> does not include all solute atoms, the neighbor list is still calculated for all solute atoms so the surface area calculated reflects the contribution of atoms in <mask> to the overall surface area, not the surface area of <mask> as an isolated system. As a result, it may be possible to obtain a negative surface area if only a small fraction of the solute is selected.

For example, to calculate the overall surface area of all solute atoms, as well as the contribution of residue 1 to the overall surface area, writing both results to "surf.dat":

```
surf out surf.dat
surf :1 out surf.dat
```

28.9.74. symmrmsd

```
symmrmsd  [<name>] [<mask>] [<refmask>] [out <filename>] [nofit] [mass] [remap]
          [ first | reference | ref <name> | refindex <#> | previous |
            reftraj <name> [parm <parmname> | parmindex <#>] ]
```

[<name>] `Output data set name.`

[<mask>] `Mask of atoms to calculate RMSD for; if not specified, calculate for all atoms.`

[<refmask>] `Reference mask; if not specified, use <mask>.`

[out <filename>] `Output data file name.`

[nofit] `Do not perform best-fit RMSD (not recommended).`

[mass] `Mass-weight the RMSD calculation.`

[remap] `Re-arrange atoms according to symmetry.  See below for more details.`

`Reference keywords:`

first `Use the first trajectory frame processed as reference.`

reference `Use the first previously read in reference structure (refindex 0).`

ref <name> `Use previously read in reference structure specified by filename/tag.`

refindex <#> `Use previously read in reference structure specified by <#> (based on order read in).`

previous `Use frame prior to current frame as reference.`

reftraj <name> `Use frames from COORDS set <name> or read in from trajectory file <name> as references. Each frame from <name> is used in turn, so that frame 1 is compared to frame 1 from <name>, frame 2 is compared to frame 2 from <name> and so on. If <trajname> runs out of frames before processing is complete, the last frame of <trajname> continues to be used as the reference.`

> **parm <parmname> | parmindex <#>** `If reftraj specifies a file associate trajectory <name> with specified topology; if not specified the first topology is used.`

Perform symmetry-corrected RMSD calculation. This is done by identifying potential symmetric atoms in each residue, performing an initial best-fit, then determining which configuration of symmetric atoms will give the lowest RMSD using atomic distance to reference atoms.

Note that when re-mapping, all atoms in the residues of interest should be selected to prevent cases where selected symmetric atoms are swapped but the atoms they are bonded to are not. Also, occasionally larger symmetric structures (e.g. 6 membered rings) may become distorted due to only part of the residue being corrected for symmetry. This appears to happen about 4% of the time but does not overly inflate the RMSD. The ***'check'*** command can be used after ***symmrmsd*** to look for such distortions.

Warning: the symmetry correction is generally robust enough to account for symmetries in the standard amino and nucleic acid residues, but has not been extensively tested on residues with more extended types of symmetry.

28.9.75. temperature

```
temperature [<name>] {frame | [<mask>] [ntc <#>]} [out <filename>]
```

[<name>] `Data set name.`

frame `Do not calculate temperature; use existing frame temperature.`

[<mask>] `Atoms to calculate temperature for.`

[ntc <#>] `Value of SHAKE bond constraint:  1 - none, 2 - bonds to H, 3- all bonds (equivalent to SANDER/PMEMD).`

[out <filename>] `File to write values to.`

Calculate temperature in frame based on velocity information. If **'frame'** is specified just use frame temperature (read in from e.g. REMD trajectory).

28.9.76. trans | translate

```
translate [<mask>] [x <dx>] [y <dy>] [z <dz>]
```

Translate atoms in **<mask>** (all atoms if no mask specified) **<dx>** Å in the X direction, **<dy>** Å in the Y direction, and **<dz>** Å in the Z direction.

28.9.77. unstrip

```
unstrip
```

Requests that the original topology and frame be used for all following actions. This has the effect of undoing any command that modifies the state (such as strip). For example, the following code takes a solvated complex and uses a combination of strip, unstrip, and outtraj commands to write out separate dry complex, receptor, and ligand files:

```
parm Complex.WAT.pdb
trajin Complex.WAT.pdb
# Remove water, write complex
strip :WAT
outtraj Complex.pdb pdb
# Reset to solvated Complex
unstrip
# Remove water and ligand, write receptor
strip :WAT,LIG
outtraj Receptor.pdb pdb
# Reset to solvated Complex
unstrip
# Remove water and receptor, write ligand
strip :WAT
strip !(:LIG)
outtraj Ligand.pdb pdb
```

28.9.78. unwrap

```
unwrap [center] [{bymol | byres | byatom}]
       [ reference | ref <name> | refindex <#> ] [<mask>]
```

[center] `Unwrap by center of mass; otherwise unwrap by first atom position.`

bymol `Unwrap by molecule (default).`

byres `Unwrap by residue.`

byatom `Unwrap by atom.`

[reference | ref <name> | refindex <#>] `Reference structure to use in unwrapping.`

[<mask>] `Selection to unwrap.`

Under periodic boundary conditions, MD trajectories are not continuous if molecules are wrapped(imaged) into the central unit cell. Especially, in sander, with *iwrap*=1, molecular trajectories become discontinuous when a molecule crosses the boundary of the unit cell. This command, **unwrap** processes the trajectories to force the *mask*ed molecules continuous by translating the molecules into the neighboring unit cells. It is the opposite function of **image**, but this command can also be used to place molecules side by side, for example, two strands of a DNA duplex. However, this command fails when the *mask*ed molecules travel more than half of the box size within a single frame.

If the optional argument "`reference`" is specified, then the first frame is unwrapped according to the reference structure. Otherwise, the first frame is not modified.

As an example, assume that :1-10 is the first strand of a DNA duplex and :11-20 is the other strand of the duplex. Then the following commands could be used to create system where the two strands are not separated artificially:

```
unwrap :1-20
center :1-20 mass origin
image origin center familiar
```

28.9.79. vector

```
vector [<name>] <Type> [out <filename> [ptrajoutput]] [<mask1>] [<mask2>]
    [magnitude] [ired]
       <Type> = { mask | minimage  | dipole | center | corrplane |
                  box  | boxcenter | ucellx | ucelly | ucellz
               principal [x|y|z] }
```

[<name>] Vector data set name.

<Type> Vector type; see below.

[out <filename>] Write vector data to <filename> with format 'Vx Vy Vz Ox Oy Oz' where V denotes vector coordinates and 'O' denotes origin coordinates.

[ptrajoutput] Write vector data in *ptraj* style (Vx Vy Vz Ox Oy Oz Vx+Ox Vy+Oy Vz+Oz). This prevents additional formatting of <filename> and is not compatible with 'magnitude'.

[<mask1>] Atom mask, required for all types except 'box'.

[<mask2>] Second atom mask, only required for type 'mask'.

[magnitude] Store the magnitude of the vector with aspect [Mag].

[ired] Mark this vector for subsequent IRED analysis with commands 'matrix ired' and 'ired'.

Data Sets Created:

<name> Vector data set.

<name>[Mag] (magnitude only) Vector magnitude.

This command will keep track of a vector value (and its origin) over the trajectory; the data can be referenced for later use based on the *name* (which must be unique). The types of vectors that can be calculated are:

mask (Default) Store vector from center of mass of atoms in **<mask1>** to atoms in **<mask2>**.

minimage Store minimum-imaged vector from center of mass of atoms in **<mask1>** to atoms in **<mask2>**.

dipole Store the dipole and center of mass of the atoms specified in **<mask1>**. The vector is not converted to appropriate units, nor is the value well-defined if the atoms in the mask are not overall charge neutral.

center Store the center of mass of atoms in **<mask1>**. The reference point is the origin (0.0, 0.0, 0.0).

corrplane This defines a vector perpendicular to the (least-squares best) plane through the atoms in **<mask1>**. The reference point is the center of mass of atoms in **<mask1>**.

box (No mask needed) Store the box lengths of the trajectory. The reference point is the origin (0.0, 0.0, 0.0).

boxcenter (No mask needed) Store the center of the box as a vector.

ucell{x|y|z}: (No mask needed) Store specified unit cell (i.e. box) vector.

principal [x|y|z] Store one of the principal axis vectors determined by diagonalization of the inertial matrix from the coordinates of the atoms specified by **<mask1>**. The eigenvector with the largest eigenvalue is considered "`x`" (i.e., the hardest axis to rotate around) and the eigenvector with the smallest eigenvalue is considered "`z`". If none of `x` or `y` or `z` are specified, then the "x" principal axis is stored. The reference point is the center of mass of atoms in **<mask1>**.

Cpptraj supports writing out vector data in a pseudo-trajectory format for easy visualization. Once a vector data set has been generated the writedata command can be used with the vectraj keyword (see 28.4 on page 561 for more details) to write a pseudo trajectory consisting of two atoms, one for the vector origin and one for the vector from the origin (i.e. V+O). For example, to create a MOL2 containing a pseudo-trajectory of the minimum-imaged vector from residue 4 to residue 11:

```
trajin tz2.nc
vector v8 minimage out v8.dat :4 :11
run
writedata v8.mol2 vectraj v8 trajfmt mol2
```

Auto-correlation or cross-correlation functions can be calculated subsequently for vectors using either the ***corr*** analysis command or the ***timecorr*** analysis command (to calculate via spherical harmonic theory).

28.9.80. velocityautocorr

```
velocityautocorr [<set name>] [<mask>] [usevelocity] [out <filename>] [diffout <file>]
                 [maxlag <frames>] [tstep <timestep>] [direct] [norm]
```

[<set name>] `Data set name.`

[<mask>] `Atoms(s) to calculate velocity autocorrelation (VAC) function for.`

[usevelocity] `Use velocity information in frame if present. This will only give sensible results if the velocities are recorded close to the order of the simulation time step.`

[out <filename>] `Write VAC function to <filename>.`

[diffout <file>] `File to write diffusion constants to.`

[maxlag <frames>] `Maximum lag in frames to calculate VAC function for. Default is half the total number of frames.`

[tstep <timestep>] `Time between frames in ps (default 1.0).`

[direct] `Calculate VAC function directly instead of via FFT (will be much slower).`

[norm] `Normalize resulting VAC function to 1.0.`

`DataSet Aspects:`

[D] Diffusion constant calculated from integral over VAC function in $1x10^{-5}$ cm^2/s.

Calculate the velocity autocorrelation (VAC) function averaged over the atoms in <mask>. Pseudo-velocities are calculated using coordinates and the specified time step. As with all time correlation functions the statistical noise will increase if the maximum lag is greater than half the total number of frames. In addition to calculating the velocity autocorrelation function, the self-diffusion coefficient will be reported in the output, calculated from the integral over the VAC function.

28.9.81. volmap

```
volmap filename dx dy dz <mask> [radscale <factor>]
    { data <existing set> |
      name <setname> { size <x,y,z> [center <x,y,z>] |
                       centermask <mask> [buffer <buffer>] } }
    [peakcut <cutoff>] [peakfile <xyzfile>]
```

filename `The name of the output file with the grid density.`

dx, dy, dz `The grid spacing (Angstroms) in the X-, Y-, and Z-dimensions, respectively`

<mask> `The atom selection from which to calculate the number density.`

radscale <factor> `Factor by which to scale radii (by division). To match the atomic radius of Oxygen used by the VMD volmap tool, a scaling factor of 1.36 should be used. Default 1.0.`

data <setname> `Name of existing grid data set to use.`

name <setname> `Name of grid set that will be created (size/center or centermask/buffer keywords).`

size <x,y,z> `Specify the size of the grid in the X-, Y-, and Z-dimensions. Must be used alongside the center argument.`

center <x,y,z> `Specify the grid center explicitly. Note, the size argument must be present in this case. Default is the origin.`

centermask <mask> `The mask around which the grid should be centered (via geometric center). If this is omitted and the center and size are not specified, the default <mask> entered (see above) is used in its place.`

buffer <buffer> `A buffer distance, in Angstroms, by which the edges of the grid should clear every atom of the centermask (or default mask if centermask is omitted) in every direction. The default value is 3. The buffer is ignored if the center and size are specified (see below).`

peakcut <cutoff> `The minimum density required to consider a local maximum a 'density peak' in the outputted peak file (default 0.05).`

peakfile <xyzfile> `A file in XYZ-format that contains a carbon atom centered at the grid point of every local density maximum. This file is necessary input to the spam action command.`

Grid data as a volumetric map, similar to the 'volmap' command in VMD. The density is calculated by treating each atom as a 3-dimensional Gaussian function whose standard deviation is equal to the van der Waals radius. The density calculated is the number density averaged over the entire simulation. The grid can be specified in one of three ways:

1. An existing grid data set (from e.g. bounds), specified with the **data** keyword.
2. Via the sizes and center specified by the **size** and **center** keywords (comma-separated strings, e.g. '20,20,20').
3. Centered on the atoms in the mask given by **centermask** with an additional buffer in each direction specified by **buffer**.

28.9.82. volume

```
volume [<name>] [out <filename>]
```

<name> `Data set name.`

out <filename> `Output file name.`

Calculate unit cell volume.

28.9.83. watershell

```
watershell <solutemask> [out <filename>] [lower <lower cut>] [upper <upper cut>]
           [noimage] [<solventmask>]
```

<solutemask> `Atom mask corresponding to solute of interest (required).`

[out <filename>] `Output file name.`

[lower <lower cut>] `Cutoff for the first water shell (default 3.4 Angstroms).`

[upper <upper cut>] `Cutoff for the second water shell (default 5.0 Angstroms).`

[noimage] `Do not image distances.`

[<solventmask>] `Optional atom mask corresponding to solvent.`

`DataSet Aspects:`

[lower] `Number of solvent molecules in first solvent shell.`

[upper] `Number of solvent molecules in second solvent shell.`

This option will count the number of waters within a certain distance of the atoms in the <solutemask> in order to represent the first and second solvation shells. The optional <solventmask> can be used to consider other atoms as the solvent; the default is “:WAT”.

This action is often used prior to the ***closest*** command in order to determine how many waters around a solute should be retained to maintain the first and/or second water shells.

28.10. Analysis Commands

Analyses in *cpptraj* operate on data sets which have been generated by Actions in a prior Run or read in with a ***readdata*** command (28.6.15 on page 573). Unlike *ptraj*, Analysis commands in *cpptraj* do not need to be prefaced with `'analysis'`. The exception to this is '***analyze matrix***' in order to differentiate it from the ***matrix*** Action command; users are encouraged to use the new command ***diagmatrix*** instead.

Like Actions, when an Analysis command is issued it is by default added to the Analysis queue and is not executed until after trajectory processing is completed; a complete list of data sets available for analysis is shown after trajectory processing (prefaced by 'DATASETS') or can be shown with the '***list dataset***' command. Analyses can also be executed immediately via the ***runanalysis*** command (28.6.19 on page 574).

Note that for Analysis commands that use COORDS data sets, if no COORDS data set is specified then a default one will be automatically created from frames read in by ***trajin*** commands.

Command	Description	Set Type(s)
autocorr	Calculate autocorrelation function for multiple data sets.	N 1D sclar
avg	Calculate average, standard deviation, min, and max for (or over) data sets.	N 1D scalar
calcstate	Calculate states based on given data sets and criteria.	N 1D scalar
cluster	Perform cluster analysis.	COORDS, N 1D scalar

corr, correlationcoe	Calculate auto or cross correlation for 1 or 2 data sets.	1D scalar, vector
crank, crankshaft	Calculate crankshaft motion between two data sets.	2 1D scalar
crdfluct	Calculate atomic fluctuations (RMSF) for atoms over time blocks.	COORDS
crosscorr	Calculate a matrix of Pearson product-moment coefficients between given data sets.	N 1D scalar
curvefit	Perform non-linear curve fitting on given data set.	1D scalar
diagmatrix	Calculate eigenvectors and eigenvalues from given symmetric matrix.	symmetric matrix
divergence	Calculate Kullback-Leibler divergence between two data sets.	2 1D scalar
FFT	Perform a fast Fourier transform on data sets.	N 1D scalar
hist, histogram	Calculate N-dimensional histogram for N given data sets.	N 1D scalar
integrate	Perform integration on each of the given data sets.	N 1D scalar
ired	Perform isotropic reoreientational eigenmode dynamics analysis using given IRED vectors.	N IRED vectors
kde	Calculate 1D histogram from given data set using a kernel density estimator. Also time-dependent Kullback-Leibler divergence analysis with another set.	1 or 2 1D scalar
lifetime	Perform lifetime analysis on given data sets.	N 1D scalar
lowestcurve	For each given data set, calculate a curve that traces the lowest N points over specified bins.	N 1D scalar
meltcurve	Calculate a melting curve from given data sets assuming simple 2 state kinetics.	N 1D scalar
modes	Perform various analyses on eigenmodes (from e.g. ***diagmatrix***).	eigenmodes
multicurve	Perform non-linear curve fitting for multiple input data sets.	N 1D scalar
multihist	Calculate 1D histograms (optionally with a kernel density estimator) from multiple input data sets.	N 1D scalar
phipsi	Calculate and plot the average phi and psi values from input dihedral data sets.	N phi/psi dihedrals
regress	Perform linear regression on multiple input data sets.	N 1D scalar
remlog	Calculate various statistics from a replica log data set.	replica log
rms2d, 2drms	Calculate 2D RMSD between frames in 1 or 2 COORDS data sets.	1 or 2 COORDS
rmsavgcorr	Calculate RMS average correlation curve for a COORDS data set.	COORDS
rotdif	Calculate rotational diffusion using given rotation matrices (from e.g. ***rms***).	rotation matrices
runningavg	Calculate running average for given data sets using given window size.	N 1D scalar
spline	Calculate cubic splines for given data sets.	N 1D scalar
stat, statistics	Calculate various statistics for given data sets.	N 1D scalar
ti	Peform Gaussian quadrature integration for given DV/DL data sets.	N 1D scalar
timecorr	Calculate auto/cross-correlation functions for given vector(s) using spherical harmonics.	1 or 2 vector
vectormath	Perform math on given vector data sets.	2 vector
wavelet	Perform wavelet analysis on coordinates from given COORDS set.	COORDS

28.10.1. autocorr

```
autocorr [name <dsetname>] <dsetarg0> [<dsetarg1> ...] [out <filename>]
         [lagmax <lag>] [nocovar] [direct]
```

<dsetarg0> [dsetarg1> ...] `Argument(s) specifying datasets to be used.`

[name <dsetname>] `Store results in dataset(s) named <dsetname>:X.`

[out <filename>] `Write results to file named <filename>.`

[lagmax] `Maximum lag to calculate for.  If not specified all frames are used.`

[nocovar] `Do not calculate covariance.`

[direct] `Do not use FFTs to calculate correlation; this will be much slower.`

This is for integer/double/float datasets only; for vectors see the ***'timecorr'*** *command.*

Calculate auto-correlation (actually auto-covariance by default) function for datasets specified by one or more dataset arguments. The datasets must have the same # of data points.

28.10.2. avg

```
avg <dset0> [<dset1> ...] [torsion] [out <file>] [oversets]
    [name <name>] [nostdout]
```

<dsetX> `Data set(s) to calculate the average for.`

[torsion] `If the data sets are not already marked periodic (e.g.  if read in via 'readdata'), treat them as periodic torsion.`

[out <file>] `File to write results to.`

[oversets] `If specified, calculate the average over all inpout sets instead of each input set.`

[name <name>] `Output data set name.`

[nostdout] `If 'nostdout' specified do not write averages to STDOUT when 'out' not specified.`

`DataSets Created (not oversets):`

<name>[avg] `Average of each set.`

<name>[sd] `Standard deviation of each set.`

<name>[ymin] `Y minimum of each set.`

<name>[ymax] `Y maximum of each set.`

<name>[yminidx] `Index of minimum Y value.`

<name>[ymaxidx] `Index of maximum Y value.`

<name>[names] `Name of each set.`

`DataSets Created (oversets)`

<name> `Average over all input sets for each frame.`

<name>[SD] `Standard deviation over all input sets for each frame.`

Calculate the average, standard deviation, min, and max of given 1D data sets. Alternatively, if **oversets** is specified the average over each set for each point is calculated; this requires all input sets be the same size.

For example, to read in data from a file named perres.peptide.dat and calculate the averages etc for all the input sets:

```
readdata perres.peptide.dat
avg perres.peptide.dat out output.dat name V
```

28.10.3. calcstate

```
calcstate {state <ID>,<dataset>,<min>,<max>} [out <state v time file>] [name <setname>]
          [curveout <curve file>] [stateout <states file>] [transout <transitions file>]
```

state <ID>,<dataset>,<min>,<max> `Define a state according to given data set and criteria.  Multiple states can be given.`

  - **<ID>** ID to give each state index (e.g. 1, 2, etc). State indices start at 0. -1 means "undefined state".
  - **<dataset>** Data set to use.
  - **<min>,<max>** Frames with data set value above <min> and below <max> will be assigned <ID>.

[out <state v time file>] File to write state index vs frame to.

[name <setname>] Data set name.

[curveout <curve file>] File to write state lifetime and transition curves to.

[stateout <states file>] File to write state lifetime data to.

[transout <transitions file>] File to write state transition data to.

DataSets Created:

<setname> State index vs frame.

<setname>[sCurve]:X State curves; lifetime curve for transitions from given state to any other state.

<setname>[tCurve]:X Transition curves; lifetime curve for transitions from given state to other specific state.

Data for the specified data set(s) that matches the given criteria will be assigned a state index. State indices start from 0 and match the order in which **state** keywords were given. For example, the following input:

```
parm DPDP.parm7
trajin DPDP.nc
distance d1 :19@O :12@N
angle a1 :19@O :12@H :12@N
calcstate state D,d1,3.0,4.0 state A,a1,100,120 out state.dat curveout curve.agr \
                stateout States.dat transout States.dat name d1_a1
run
```

Defines two states. State index 0 is defined as a state named "D" based on the distance from ':19@O' to ':12@N' being between 3 and 4 Angstroms. State index 1 is defined as a state named "A" based on the angle between ':19@O', ':12@H', and ':12@N' being between 100 and 120 degrees. The output in state.dat might look like:

```
#Frame          d1_a1
       1           -1
       2            0
       3            0
       4            0
       5           -1
       6            1
       7           -1
       8           -1
       9            0
      10           -1
```

where the values in column d1_a1 refer to state index: -1 is undefined, 0 is state "D", and 1 is state "A".

Lifetime curves (see 28.10.17 on page 665 for further explanation) are calculated for transitions from each state to any other state (aspect [sCurve]) and each state to each other state (aspect [tCurve]). In this case there will be 3 sCurves and 4 tCurves:

```
d1_a1[sCurve]:0 "Undefined" (double), size is 10
d1_a1[sCurve]:1 "D" (double), size is 3
d1_a1[sCurve]:2 "A" (double), size is 1
```

```
d1_a1[tCurve]:0 "Undefined->D" (double), size is 10
d1_a1[tCurve]:1 "D->Undefined" (double), size is 3
d1_a1[tCurve]:2 "Undefined->A" (double), size is 1
d1_a1[tCurve]:3 "A->Undefined" (double), size is 1
```

Lifetime analysis from each state to any other state is directed to the file specified by **stateout** and has format:

```
#Index N Average Max State
```

Where `#Index` is the state index, `N` is the number of lifetimes in that state, `Average` is the average lifetime while in that state (in frames), `Max` is the maximum lifetime while in that state (in frames) and `State` is the name of the state.

Finally, lifetime analysis of transitions from each state to each other state is directory to the file specified by transout and has format:

```
#N Average Max Transition
```

Where `#N` is the number of transitions, `Average` is the average lifetime (in frames) in the first state before transitioning to the second state, `Max` is the max lifetime (in frames) before transitioning to the second state, and `Transition` is the name of the transition.

28.10.4. cluster

```
cluster [crdset <crd set> | nocoords]
  Algorithms:
[hieragglo [epsilon <e>] [clusters <n>] [linkage|averagelinkage|complete]
      [epsilonplot <file>] [includesieved_cdist]]
[dbscan minpoints <n> epsilon <e> [sievetoframe] [kdist <k> [kfile <prefix>]]]
    [dpeaks epsilon <e> [noise] [dvdfile <density_vs_dist_file>]
  [choosepoints {manual | auto}]
  [distancecut <distcut>] [densitycut <densitycut>]
  [runavg <runavg_file>] [deltafile <file>] [gauss]]
    [kmeans clusters <n> [randompoint [kseed <seed>]] [maxit <iterations>]
    [{readtxt|readinfo} infofile <file>]
  Distance options:
{[[rms | srmsd] [<mask>] [mass] [nofit]] | [dme [<mask>]] |
      [data <dset0>[,<dset1>,...]]}
[sieve <#> [random [sieveseed <#>]]] [loadpairdist] [savepairdist] [pairdist <file>]
    [pairwisecache {mem | none}] [includesieveincalc]
  Output options:
[out <cnumvtime>] [gracecolor] [summary <summaryfile>] [info <infofile>]
[summarysplit <splitfile>] [splitframe <comma-separated frame list>]
    [clustersvtime <filename> cvtwindow <window size>]
[cpopvtime <file> [normpop | normframe] [lifetime]]
    [sil <silhouette file prefix>] [assignrefs [refcut <rms>] [refmask <mask>]]
  Coordinate output options:
[ clusterout <trajfileprefix> [clusterfmt <trajformat>] ]
[ singlerepout <trajfilename> [singlerepfmt <trajformat>] ]
[ repout <repprefix> [repfmt <repfmt>] [repframe] ]
    [ avgout <avgprefix> [avgfmt <avgfmt>] ]
```

[crdset <crd set>] `Name of previously generated COORDS data set. If not specified the default COORDS set will be used unless nocoords has been specified.`

[nocoords] `Do not use a COORDS data set; distance metrics that require coordinates and coordiante output will be disabled.`

`Algorithms:`

hieragglo `(Default) Use hierarchical agglomerative (bottom-up) approach.`

- **[epsilon <e>]** `Finish clustering when minimum distance between clusters is greater than <e>.`
- **[clusters <n>]** `Finish clustering when <n> clusters remain.`
- **[linkage]** `Single-linkage; use the shortest distance between members of two clusters.`
- **[averagelinkage]** `Average-linkage (default); use the average distance between members of two clusters.`
- **[complete]** `Complete-linkage; use the maximum distance between members of two clusters.`
- **[epsilonplot <file>]** `Write number of clusters vs epsilon to <file>.`
- **[includesieved_cdist]** `Include sieved frames in final cluster distance calculation (may be very slow).`

dbscan `Use DBSCAN clustering algorithm of Ester et al.`[577]

- **minpoints <n>** `Minimum number of points required to form a cluster.`
- **epsilon <e>** `Distance cutoff between points for forming a cluster.`
- **[sievetoframe]** `When restoring sieved frames, compare frame to every frame in a cluster instead of the centroid; slower but more accurate.`
- **[kdist <k>]** `Generate K-dist plot for help in determining DBSCAN parameters (see below).`
- **[kfile <prefix>]** `Prefix for K-dist plot file.`

dpeaks `Use the density peaks algorithm of Rodriguez and Laio`[578]

- **epsilon <e>** `Cutoff for determining local density in Angstroms.`
- **[noise]** `If specified, treat all points within epsilon of another cluster as noise.`
- **[dvdfile <density_vs_dist_file>]** `File to write density versus minimum distance to point with next highest density. This can be used to determine appropriate cutoffs for distance and density in a subsequent step with choosepoints manual.`
- **[choosepoints {manual | auto}]** `Specify whether clusters will be chosen based on specified distance/density cutoffs, or automatically. If not specified only the density vs distance file will be written and no clustering will be performed. Currently manual is recommended.`
- **[distancecut <distcut>] [densitycut <densitycut>]** `If choosepoints manual, points with minimum distance greather than or equal to <distcut> and density greater than or equal to <densitycut> will be chosen.`
- **[runavg <runavg file>]** `If choosepoints automatic, the calculated running average of density versus distance will be written to <runavg file>.`
- **[deltafile <file>]** `If choosepoints automatic, distance minus the running average for each point will be written to this file.`
- **[gauss]** `Calculate density with Gaussian kernels instead of using discrete density.`

kmeans `Use K-means clustering algorithm.`

- **clusters <n>** `Finish clustering when number of clusters is <n>.`
- **[randompoint]** `Randomize initial set of points used (recommended).`
- **[kseed <seed>]** `Random number generator seed for randompoint.`

[maxit <iteration>] Algorithm will run until frames no longer change clusters of <iteration> iterations are reached (default 100).

readtxt|readinfo No clustering - read in previous cluster results.

infofile <file> Cluster info file to read.

Distance Metric Options:

[rms | srmsd[<mask>]] (Default rms) Distance between frames calculated via best-fit coordinate RMSD using atoms in <mask>. If srmsd specified use symmetry-corrected RMSD (see 28.9.74 on page 642).

[mass] Mass-weight the RMSD.

[nofit] Do not fit structures onto each other prior to calculating RMSD.

dme [<mask>] Distance between frames calculated using distance-RMSD (aka DME, *distrmsd*) using atoms in <mask>.

[data <dset0>[,<dset1>,...] Distance between frames calculated using specified data set(s) (Euclidean distance).

[sieve <#>] Perform clustering only for every <#> frame. After clustering, all other frames will be added to clusters.

[random] When sieve is specified, select initial frames to cluster randomly.

[sieveseed <#>] Seed for random sieving; if not set the wallclock time will be used.

[pairdist <file>] File to use for loading/saving pairwise distances.

[loadpairdist] Load pairwise distances from <file> (CpptrajPairDist if pairdist not specified).

[savepairdist] Save pairwise distances from <file> (CpptrajPairDist if pairdist not specified). NOTE: If sieving was performed only the calculated distances are saved.

[pairwisecache {mem | disk | none}] Cache pairwise distance data in memory (default), to disk, or disable pairwise caching. No caching will save memory but be extremely slow. Caching to disk will likely be slow unless writing to a fast storage device (e.g. SSD) - data is saved to a file named 'CpptrajPairwiseCache'.

[includesieveincalc] Include sieved frames when calculating within-cluster average (may be very slow).

Output Options:

[out <cnumvtime>] Write cluster # vs frame to <cnumvtime>. Algorithms that calculate noise (e.g. DBSCAN) will assign noise points a value of -1.

[gracecolor] Instead of cluster # vs frame, write cluster# + 1 (corresponding to colors used by XMGRACE) vs frame. Cluster #s larger than 15 are given the same color. Algorithms that calculate noise (e.g. DBSCAN) will assign noise points a color of 0 (blank).

[summary <summaryfile>] Summarize each cluster with format '#Cluster Frames Frac AvgDist Stdev Centroid AvgCDist':

#Cluster Cluster number starting from 0 (0 is most populated).

Frames # of frames in cluster.

Frac Size of cluster as fraction of total trajectory.

AvgDist Average distance between points in the cluster.

Stdev Standard deviation of points in the cluster.

Centroid Frame # of structure in cluster that has the lowest cumulative distance to every other point.

AvgCDist Average distance of this cluster to every other cluster.

[info <infofile>] Write ptraj-like cluster information to <infofile>. This file has format:

```
#Clustering: <X> clusters <N> frames
#Cluster <I> has average-distance-to-centroid <AVG>
...
#DBI: <DBI>
#pSF: <PSF>
#Algorithm: <algorithm-specific info>
<Line for cluster 0>
...
#Representative frames: <representative frame list>
```

Where <X> is the number of clusters, <N> is the number of frames clustered, <I> ranges from 0 to <X>-1, <AVG> is the average distance of all frames in that cluster to the centroid, <DBI> is the Davies-Bouldin Index, <pSF> is the pseudo-F statistic, and <representative frame list> contains the frame # of the representative frame (i.e. closest to the centroid) for each cluster. Each cluster has a line made up of characters (one for each frame) where '.' means 'not in cluster' and 'X' means 'in cluster'.

[summarysplit <splitfile>] Summarize each cluster based on which of its frames fall in portions of the trajectory specified by splitframe with format '#Cluster Total Frac C# Color NumInX ... FracX ... FirstX':

#Cluster Cluster number starting from 0 (0 is most populated).

Total # of frames in cluster.

Frac Size of cluster as a fraction of the total trajectory.

C# Grace color number.

Color Text description of the color (based on standard XMGRACE coloring).

NumInX Number of frames in Xth portion of the trajectory.

FracX Fraction of frames in Xth portion of the trajectory.

FirstX Frame in the Xth portion of the trajectory where the cluster is first observed.

[splitframe <frame>] For summarysplit, frame or comma-separated list of frames to split the trajectory at, e.g. '100,200,300'.

[bestrep {cumulative|centroid|cumulative_nosieve}] Method for choosing cluster representative frames.

cumulative Choose by lowest cumulative distance to all other frames in cluster. Default when not sieving.

centroid Choose by lowest distance to cluster centroid. Default when sieving.

cumulative_nosieve Choose by lowest cumulative distance to all other frames, ignoring sieved frames.

[clustersvtime <filename>] Write number of unique clusters observed in a given time window to <filename>.

[cvtwindow <windowsize>] Window size for clustersvtime output.

[cpopvtime <file> [normpop | normframe]] Write cluster population vs time to <file>; if normpop specified normalize each cluster to 1.0; if normframe specified normalize cluster populations by number of frames.

[sil <prefix>] `Write average cluster silhouette value for each cluster to '<prefix>.cluster.dat' and cluster silhouette value for each individual frame to '<prefix>.frame.dat'.`

assignrefs `In summary/summarysplit, assign clusters to loaded representative structures if RMSD to that reference is less than specified cutoff.`

[refcut <rms>] `RMSD cutoff in Angstroms.`

[refmask <mask>] `Mask to use for RMSD calculation.  If not specified the default mask is all heavy atoms.`

`Coordinate Output Options:`

clusterout <trajfileprefix> `Write frames in each cluster to files named <trajfileprefix>.cX, where X is the cluster number.`

clusterfmt <trajformat> `Format keyword for clusterout (default Amber Trajectory).`

singlerepout <trajfilename> `Write all representative frames to single trajectory named <trajfilename>.`

singlerepfmt <trajformat> `Format keyword for singlerepout (default Amber Trajectory).`

repout <repprefix> `Write representative frames to separate files named <repprefix>.X.<ext>, where X is the cluster number and <ext> is a format-specific filename extension.`

repfmt <trajformat> `Format keyword for repout (default Amber Trajectory).`

repframe `Include representative frame number in repout filename.`

avgout <avgprefix> `Write average structure for each cluster to separate files named <avgprefix>.X.<ext>, where X is the cluster number and <ext> is a format-specific filename extension.`

avgfmt <trajformat> `Format keyword for avgout.`

`DataSet Aspects:`

[Pop] `Cluster population vs time; index corresponds to cluster number.`

Note cluster population vs time data sets are not generated until the analysis has been run.

Cluster input frames using the specified clustering algorithm and distance metric. In order to speed up clustering of large trajectories, the **sieve** keyword can be used. In addition, subsequent clustering calculations can be sped up by writing/reading calculated pair distances between each frame to/from a file specified by **pairdist** (or "CpptrajPairDist" if **pairdist** not specified).

Example: cluster on a specific distance:

```
distance endToEnd :1 :255
cluster data endToEnd clusters 10 epsilon 3.0 summary summary.dat info info.dat
```

Example: cluster on the CA atoms of residues 2-10 using average-linkage, stopping when either 3 clusters are reached or the minimum distance between clusters is 4.0, writing the cluster number vs time to "cnumvtime.dat" and a summary of each cluster to "avg.summary.dat":

```
cluster C1 :2-10 clusters 3 epsilon 4.0 out cnumvtime.dat summary avg.summary.dat
```

Clustering Metrics

The Davies-Bouldin Index (DBI) measures sum over all clusters of the within cluster scatter to the between cluster separation; **the smaller the DBI, the better**. The DBI is defined as the average, for all clusters X, of fred, where fred(X) = max, across other clusters Y, of (Cx + Cy)/dXY. Here Cx is the average distance from points in X to the centroid, similarly Cy, and dXY is the distance between cluster centroids.

The pseudo-F statistic (pSF) is another measure of clustering goodness. It is intended to capture the 'tightness' of clusters, and is in essence a ratio of the mean sum of squares between groups to the mean sum of squares within group. **High values are good**. Generally, one selects a cluster-count that gives a peak in the pseudo-f statistic. Formula: A/B, where A = (T - P)/(G-1), and B = P / (n-G). Here n is the number of points, G is the number of clusters, T is the total distance from the all-data centroid, and P is the sum (for all clusters) of the distances from the cluster centroid.

The cluster silhouette is a measure of how well each point fits within a cluster. Values of 1 indicate the point is very similar to other points in the cluster, i.e. it is well-clustered. Values of -1 indicate the point is dissimilar and may fit better in a neighboring cluster. Values of 0 indicate the point is on a border between two clusters.

Hints for setting DBSCAN parameters with 'kdist'

It is not always obvious what parameters to set for DBSCAN. You can get a rough idea of what to set 'mindist' and 'epsilon' to by generating a so-called "K-dist" plot with the 'kidst <k>' option. The K-dist plot shows for each point (X axis) the Kth farthest distance (Y axis), sorted by decreasing distance. You supply the same distance metric and sieve parameters you want to use for the actual clustering, but nothing else. For example:

```
cluster C0 dbscan kdist 4 rms :1-4@CA sieve 10 loadpairdist pairdist CpptrajPairDist
```

The K-dist plot will be named <prefix>.<k>.dat, with the default prefix being 'Kdist' (in this case the file name would be Kdist.4.dat). The K-dist plot usually looks like a curve with an initially steep slope that gradually decreases. Around where the initial part of the curve starts to flatten out (indicating an increas in density) is around where epsilon should be set; minpoints is set to whatever <k> was. It has been suggested that the shape of the K-dist curve doesn't change too much after Kdist=4, but users are encouraged to experiment.

Using 'dpeaks' clustering

The 'dpeaks' (density peaks) algorithm attempts to find clusters by identifying points in high density regions which are far from other points of high density[578]. There are two ways these points can be chosen. The first and recommended way is manually. In this method, clustering if first run with **choosepoints** not specified to generate a plot containing density versus minimum distance to point with next highest density (the decision graph). Appropriate cut offs for distance and density can then be chosen based on visual inspection; cutoffs should be chosen so that they select points that have both a high density and a high distance to point with next highest density. Clustering can then be run again with **distancecut** and **densitycut** set.

The second way is automatically; cpptraj will attempt to identify outliers in the density vs distance plot based on distance from the running average. Although this only requires a single pass, this method of choosing points is not well-tested and currently not recommended.

The CpptrajPairDist file format

The CpptrajPairDist file is binary; the exact format depends on what version of cpptraj generated the file (since earlier versions had no concept of 'sieve'). The CpptrajPairDist file starts with a 4 byte header containing the characters 'C' 'T' 'M' followed by the version number. A quick way to figure out the version is to use the linux 'od' command to output the first 4 bytes as hexadecimal, e.g.:

```
$ od -t x1 -N 4 CpptrajPairDist 0000000 43 54 4d 02
```

So the CpptrajPairDist file version in the above example is 2.

The next few numbers describe the matrix size and depend on the version.

Version 0: Two 4-byte integers: # of rows and # of elements.

Version 1: Two 8-byte unsigned integers (equivalent to size_t on most systems): # of rows and # of elements.

Version 2: Three 8 byte unsigned integers: original # of rows, actual # of rows, and sieve value.

This is followed by the actual matrix data, stored as a single array of floats (4 bytes). For versions 1 and 2 the number of elements is explicitly stored. For version 2, to calculate the number of matrix elements you need to read:

```
Elements = (actual_rows * (actual_rows - 1)) / 2
```

The cluster pair-distance matrix is an upper-right triangle matrix without the diagonal (in row-major order), so the first element is the distance between elements 0 and 1, the second is between elements 0 and 2, etc.

In version 2 files, if the sieve value is greater than 1 that means original_rows > actual_rows and there is an additional array of characters original_nrows long, with 'T' if the row is being ignored (i.e. it was sieved out) and 'F' if the row is active (i.e. is active in the actual pairwise-distance matrix).

The code that cpptraj uses to read in CpptrajPairDist files is in ClusterMatrix::LoadFile() (ClusterMatrix.cpp).

28.10.5. corr | correlationcoe

```
corr out <outfilename> <dataset1> [<dataset2>]
     [lagmax <lag>] [nocovar] [direct]
```

out <outfilename> `Write results to file named <outfilename>. The datasets must have the same # of data points.`

<dataset1> [<dataset2>] `Data set(s) to calculate correlation for. If one dataset or the same dataset is given twice, the auto-correlation will be calculated, otherwise cross-correlation.`

[lagmax] `Maximum lag to calculate for. If not specified all frames are used.`

[nocovar] `Do not calculate covariance.`

[direct] `Do not use FFTs to calculate correlation; this will be much slower.`

`DataSet Aspects:`

[<dataset1>] `(Auto-correlation) The aspect will be the name of each of the input data set.`

[<dataset1>-<dataset2>] `(Cross-correlation) The aspect will be the names of each of the input data sets joined by a dash ('-').`

`DataSet Aspects:`

[coeff] `Correlation coefficient.`

Calculate the auto-correlation function for data set named <dataset1> or the cross-correlation function for data sets named <dataset1> and <dataset2> up to <lagmax> frames (all if **lagmax** not specified), writing the result to file specified by **out**. The two datasets must have the same # of datapoints.

28.10.6. crank | crankshaft

```
crank {angle | distance} <dsetname1> <dsetname2> info <string>
      [out <filename>] [results <resultsfile>]
```

angle `Analyze angle data sets.`

distance `Analyze distance data sets.`

<dsetname1> `Data set to analyze.`

<dsetname2> `Data set to analyze.`

info <string> `Title the analysis <string>.`

[out <filename>] `Write frame-vs-bin to <filename>.`

[results <resultsfile>] `Write results to <resultsfile>.`

Calculate crankshaft motion between two data sets.

28.10.7. crdfluct

```
[crdset <crd set>] [<mask>] [out <filename>] [window <size>] [bfactor]
```

Calculate atomic positional fluctuations for atoms in **<mask>** over windows of size **<size>**. If **bfactor** is specified, the fluctuations are weighted by $\frac{8}{3}\pi^2$(similar but not necessarily equivalent to crystallographic B-factor calculation). Units are Å, or Å^{2}x$\frac{8}{3}\pi^2$ if **bfactor** specified.

28.10.8. crosscorr

```
crosscorr [name <dsetname>] <dsetarg0> [<dsetarg1> ...] [out <filename>]
```

[name <dsetname>] The resulting upper-triangle matrix is stored with name <dsetname>.

<dsetarg0> [<dsetarg1> ...] Argument(s) specifying datasets to be used.

[out <filename>] Write results to file named <filename>.

Calculate the Pearson product-moment correlation coefficients between all specified datasets.

28.10.9. curvefit

```
curvefit <dset> { <equation>  |
                  name <dsname> { gauss | nexp <m> [form {mexp|mexpk|mexpk_penalty} } }
         [AX=<value> ...] [out <outfile>] [resultsout <results>]
         [maxit <max iterations>] [tol <tolerance>]
         [outxbins <NX> outxmin <xmin> outxmax <xmax>]
```

<dset> Data set to fit.

<equation> Equation to fit of form <Variable> = <Equation>. See 28.3.2 on page 559 for more details on equations *cpptraj* understands.

name <dsname> Final data set name (required if using nexp or gauss).

gauss Fit to Gaussian of form A0 * exp(-((X - A1)^2) / (2 * A2^2))

nexp <m> Fit to specified number of exponentials.

- **form <type>** Fit to specified exponential form:
  - **mexp** Multi-exponential, SUM(m)[An * exp(An+1 * X)]
  - **mexpk** Multi-exponential plus constant, A0 + SUM(m)[An * exp(An+1 * X)]
  - **mexpk_penalty** Same as mexpk except sum of prefactors constrained to 1.0 and exponential constants constrained to < 0.0.

AX=<value> Value of any constants in specified equation with X starting from 0 (can specify more than one).

out <outfile> Write resulting fit curve to <outfile>.

resultsout <results> Write details of the fit to <results> (default STDOUT).

maxit <max iterations> Number of iterations to run curve fitting algotrithm (default 50).

tol <tolerance> Curve-fitting tolerance (default 1E-4).

outxbins <NX> Number of points to use when generating final curve (default same number of points as input data set).

outxmin <xmin> Minimum X value to use for final curve (default same number of points as input data set).

outxmax <xmax> Maximum X value to use for final curve (default same number of points as input data set).

Perform non-linear curve fitting for the specified data set using the Levenberg-Marquardt algorithm. Any equation form that cpptraj understands (see 28.3.2 on page 559) can be used, or several preset forms can be used. Similar to Grace (http://plasma-gate.weizmann.ac.il/Grace/), an equation can contain constants for curve fitting termed AX (with X being a numerical digit, one for each constant), and is assigned to a variable which then becomes a data set. For example, to fit a curve to data from a file named Data.dat to a data set named 'FitY':

```
readdata Data.dat
runanalysis curvefit Data.dat \
  "FitY = (A0 * exp(X * A1)) + (A2 * exp(X * A3))" \
  A0=1 A1=-1 A2=1 A3=-1 \
  out curve.dat tol 0.0001 maxit 50
```

To perform the same fit but to a multi-exponential curve with two exponentials:

```
readdata Data.dat
runanalysis curvefit Data.dat nexp 2 name FitY \
  A0=1 A1=-1 A2=1 A3=-1 \
  out curve1.dat tol 0.0001 maxit 50
```

28.10.10. diagmatrix

```
diagmatrix <name> [out <filename>] [thermo [outthermo <filename>]]
           [vecs <#>] [name <modesname>] [reduce]
           [nmwiz [nmwizvecs <#>] [nmwizfile <filename>]]
```

<name> Name of symmetric matrix to diagonalize.

[out <filename>] Write results to <filename>.

[thermo [outthermo <filename>]] Mass-weighted covariance (mwcovar) matrix only. Calculate entropy, heat capacity, and internal energy from the structure of a molecule (average coordinates, see above) and its vibrational frequencies using standard statistical mechanical formulas for an ideal gas. Results are written to <filename> if specified, otherwise results are written to STDOUT. Note that this converts the units of the calculated eigenvalues to frequencies (cm^{-1}).

[vecs <#>] Number of eigenvectors to calculate. Default is 0, which is only allowed when 'thermo' is specified.

[name <modesname>] Store resulting modes data set with name <modesname>.

[reduce] Covariance (covar/mwcovar/distcovar) matrices only. For coordinate covariance (covar/mwcovar) matrices, each eigenvector element is reduced via Ei = Eix^2 + Eiy^2 + Eiz^2. For distance covariance (distcovar) the eigenvectors are reduced by taking the sum of the squares of each row. See Abseher & Nilges, JMB 1998, 279, 911-920 for further details. They may be used to compare results from PCA in distance space with those from PCA in cartesian-coordinate space.

[nmwiz] Generate output in *.nmd* format file for viewing with NMWiz[579]. See http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/ for further details.

- **[nmwizvecs <#>]** Number of vectors to write out for nmwiz output, starting with the lowest frequency mode (default 20).
- **[nmwizfile <filename>]** Name of nmwiz file to write to (default 'out.nmd').

[nmwizmask <mask>] `Mask of atoms corresponding to eigenvectors - should be the same one used to generate the matrix.`

Calculate eigenvectors and eigenvalues for the specified symmetric matrix. This is followed by Principal Component Analysis (in cartesian coordinate space in the case of a covariance matrix or in distance space in the case of a distance-covariance matrix), or Quasiharmonic Analysis (in the case of a mass-weighted covariance matrix). Diagonalization of distance, correlation, idea, and ired matrices are also possible. Eigenvalues are given in cm^{-1} in the case of a mass-weighted covariance matrix and in the units of the matrix elements in all other cases. In the case of a mass-weighted covariance matrix, the eigenvectors are mass-weighted.

Note that the maximum number of non-zero eigenvalues obtainable depends on the number of frames used to generate the input matrix; the number of frames should be equal to or greater than the number of columns in the matrix in order to obtain all eigenmodes.

Results may include average coordinates (in the case of covar, mwcovar, correl), average distances (in the case of distcovar), main diagonal elements (in the case of idea and ired), eigenvalues, and eigenvectors.

For example, in the following a mass-weighted covariance matrix of all atoms is generated and stored internally with the name mwcvmat; the matrix itself is written to mwcvmat.dat. Subsequently, the first 20 eigenmodes of the matrix are calculated and written to evecs.dat, and quasiharmonic analysis is performed at 300.0 K, with the results written to thermo.dat.

```
matrix mwcovar name mwcvmat out mwcvmat.dat
diagmatrix mwcvmat out evecs.dat vecs 20 \
           thermo outthermo thermo.dat temp 300.0
```

Output Format

The “modes” or “evecs” output file is a text file with the following format:

```
 [Reduced] Eigenvector file: <Type> nmodes <#> width <width>
  <# Avg Coords> <Eigenvector Size>
<Average Coordinates>
```

Where <Type> is a string identifying what kind of matrix the eigenvectors/eigenvalues were determined from, nmodes is how many eigenvectors are in the file, and <Average Coordinates> are in lines 7 columns wide, with each element having width specified by <width>. Then for each eigenvector:

```
 ****
<Eigenvector#> <Eigenvalue>
<Eigenvector Coordinates>
...
```

Where <Eigenvector Coordinates> are in lines 7 columns wide, with each element having width specified by <width>.

28.10.11. divergence

```
divergence ds1 <ds1> ds2 <ds2>
```

Calculate Kullback-Leibler divergence between specified data sets.

28.10.12. fft

```
fft <dset0> [<dset1> ...] [out <outfile>] [name <outsetname>] [dt <samp_int>]
```

<dset0> [<dset1 ...] `Argument(s) specifying datasets to be used.`

[out <outfile>] `Write results to file named <outfile>.`

[name <outsetname>] The resulting transform will be stored with name <outsetname>.

[dt <samp_int>] Set the sampling interval (default is 1.0).

Perform fast Fourier transform (FFT) on specified data set(s). If more than 1 data set, they must all have the same size.

28.10.13. hist | histogram

```
hist <dataset_name>[,<min>,<max>,<step>,<bins>] ...
[free <temperature>] [norm | normint] [gnu] [circular] out <filename>
[amd <amdboost_data>] [name <outputset name>]
    [traj3d <file> [trajfmt <format>] [parmout <file>]]
[min <min>] [max <max>] [step <step>] [bins <bins>] [nativeout]
```

<dataset_name>[,<min>,<max>,<step>,<bins>] Dataset(s) to be histogrammed. Optionally, the min, max, step, and/or number of bins can be specified for this dimension after the dataset name separated by commas. It is only necessary to specify the step or number of bins, an asterisk '*' indicates the value should be calculated from available data.

[free <temperature>] If specified, estimate free energy from bin populations using $G_i = -k_B T \ln\left(\frac{N_i}{N_{Max}}\right)$, where K_B is Boltzmann's constant, T is the temperature specified by <temperature>, N_i is the population of bin i and N_{Max} is the population of the most populated bin. Bins with no population are given an artificial barrier equivalent to a population of 0.5.

[norm] If specified, normalize bin populations so the sum over all bins equals 1.0.

[normint] Normalize bin populations so the integral over them is 1.0.

[gnu] Internal output only; data will be gnuplot-readable, i.e. a space will be printed after the highest order coordinate cycles.

[circular] Internal output only; data will wrap, i.e. an extra bin will be printed before min and after max in each direction. Useful for e.g. dihedral angles.

out <filename> Write results to file named <filename>.

[amd <amdboost_data>] Reweight bins using AMD boost energies in data set <amdboost_data> (in KT).

[name <outputset name>] Output histogram data set name.

[traj3d <file> [trajfmt <format>]] (3D histograms only) Write a pseudo-trajectory of the 3 data sets (1 atom) to <file> with format <format>.

[parmout <file>] (3D histograms only) Write a topology corresponding to the pseudo-trajectory to <file>.

[min <min>] Default minimum to bin if not specified.

[max <max>] Default max to use if not specified.

[step <step>] Default step size to use if not specified.

[bins <bins>] Default bin size to use if not specified.

[nativeout] Do not use cpptraj data file framework; only necessary for writing out histograms with > 3 dimensions.

Create an N-dimensional histrogram, where N is the number of datasets specified. For 1-dimensional histograms the xmgrace '.agr' file format is recommended; for 2-dimensional hisograms the gnuplot '.gnu' file format is recommended; for all other dimensions plot formatting is disabled and the routine uses its own internal output format; this is also enabled if **gnu** or **circular** is specified.

For example, to create a two dimensional histogram of two datasets 'phi' and 'psi':

```
dihedral phi :2@C :3@N :3@CA :3@C
dihedral psi :3@N :3@CA :3@C :4@N
hist phi,-180,180,*,72 psi,-180,180,*,72 out hist.gnu
```

In this case the number of bins (72) has been specified for each dimension and '*' has been given for the step size, indicating it should be calculated based on min/max/bins. The following 'hist' command is equivalent:

```
hist phi psi min -180 max 180 bins 72 out hist.gnu
```

28.10.14. integrate

```
integrate <dset0> [<dset1> ...] [out <outfile>] [name <outsetname>]
```

Integrate specified data set(s) using trapezoid integration.

28.10.15. ired

```
ired [relax freq <MHz> [NHdist <distnh>]] [order <order>]
  tstep <tstep> tcorr <tcorr> out <filename> [norm] [drct]
  modes <modesname> [name <output sets name>] [ds2matrix <file>]
```

[relax freq <MHz> [NHdist <distnh>]] `Should only be used when ired vectors represent N-H bonds; calculate correlation times` τ_m `for each eigenmode and relaxation rates and NOEs for each N-H vector. 'freq <MHz>' (required) is the Lamor frequency of the measurement. 'NHdist <distnh>' specifies the length of the NH bond in Angstroms (default is 1.02).`

order <order> `Order of the Legendre polynomials to use when calculating spherical harmonics (default 2).`

tstep <tstep> `Time between snapshots in ps (default 1.0).`

tcorr <tcorr> `Maximum time to calculate correlation functions for in ps (default 10000.0).`

out <filename> `Name of file to write output to.`

[norm] `Normalize all correlation functions, i.e.,` $C_l(t=0) = P_l(t=0) = 1.0$.

[drct] `Use the direct method to calculate correlations instead of FFT; this will be much slower.`

modes <modesname> `Name of previously calculated eigenmodes corresponding to IRED vectors.`

[name <name>] `Output data set name.`

[ds2matrix <file>] `If specified, write full delta*S^2 matrix (# IRED vector rows by # eigenmodes columns) to <file>.`

`DataSets Created:`

<name>[S2] `S2 order parameters for each vector.`

<name>[Plateau] `Plateau values for each vector.`

<name>[TauM] `TauM values for each vector.`

<name>[dS2] `Full delta*S^2 matrix.`

<name>[T1] `T1 relaxation values for each vector.`

<name>[T2] `T2 relaxation values for each vector.`

<name>[NOE] `NOEs for each vector.`

<name>[Cm(t)]:X `Cm(t) function for vector X.`

<name>[Cj(t)]:X `Cj(t) function for vector X.`

Peform IRED[566] analysis on previously defined IRED vectors (see vector ired) using eigenmodes calculated from those vectors with a previous 'diagmatrix' command. The number of defined IRED vectors should match the number of eigenmodes calculated. Autocorrelation functions for each mode and the corresponding correlation time τ_m will be written to *filename.cmt*. Autocorrelation functions for each vector will be written to *filename.cjt*. Relaxation rates and NOEs for each N-H vector will be written to <filename> or added to the the end of the standard output. For the calculation of τ_m the normalized correlation functions and only the first third of the analyzed time steps will be used. For further information on the convergence of correlation functions see [Schneider, Brünger, Nilges, *J. Mol. Biol.* **285**, 727 (1999)].

Example of IRED in Cpptraj

In *cpptraj*, IRED analysis[566] can now be performed in one pass (as opposed to the two passes previously required in *ptraj*). First, IRED vectors are defined (in this case for N-H bonds) and an IRED matrix is calculated and analyzed. The IRED vectors are then projected onto the calculated IRED eigenvectors in the ***ired*** analysis command to calculate the time correlation functions. If the parameter *order* is specified, order parameters based on IRED are calculated. By specifying the *relax* parameter, relaxation rates and NOEs can be obtained for each N-H vector. Note that the order of the IRED matrix should be the same as the one specified for IRED analysis.

```
# Define N-H IRED vectors
vector v0 @5 ired @6
vector v1 @7 ired @8
...
vector v5 @15 ired @16
vector v6 @17 ired @18`
# Define IRED matrix using all previous IRED vectors
matrix ired name matired order 2
# Diagonalize IRED matrix
diagmatrix matired vecs 6 out ired.vec name ired.vec
# Perform IRED analysis
ired relax NHdist 1.02 freq 500.0 tstep 1.0 tcorr 100.0 out v0.out noefile noe order 2
```

28.10.16. kde

```
kde <dataset> [bandwidth <bw>] [out <file>] [name <dsname>]
    [min <min>] [max <max>] [step <step>] [bins <bins>] [free]
    [kldiv <dsname2> [klout <outfile>]] [amd <amdboost_data>]
```

[bandwidth <bw>] `Bandwidth to use for KDE; if not specified bandwidth will be estimated using the normal distribution approximation.`

[out <file>] `Output file name.`

[name <dsname>] `Output data set name.`

[min <min>] `Minimum bin.`

[max <max>] `Maximum bin.`

[step <step>] `Bin step.`

[bins <bins>] `Number of bins.`

[free] `Calculate free energy from bin population.`

[kldiv <dsname2> [klout <outfile>]] `Calculate Kullback-Leibler divergence over time of <dataset> distribution to <dsname2> distribution. Output to <outfile> if klout specified.`

[amd <amdboost_data>] `Reweight histogram using AMD boost data from data set <amdboost_data> (in KT).`

Histogram 1D data set using a Gaussian kernel density estimator.

28.10.17. lifetime

```
lifetime [out <filename>] <dsetarg0> [ <dsetarg1> ... ]
      [window <windowsize> [name <setname>]] [averageonly]
      [cumulative] [delta] [cut <cutoff>] [greater | less] [rawcurve]
         [fuzz <fuzzcut>] [nosort]
```

[out <filename>] `Write results to file named <filename>, and lifetime curves to 'crv.<filename>'. If performing windowed lifetime analysis, <filename> contains the fraction present over time windows, and 2 additional files are written: 'max.<filename>', containing max lifetime over windows, and 'avg.<filename>', containing average lifetime over windows.`

<dsetarg0> [<dsetarg1> ...] `Argument(s) specifying datasets to be used.`

[window <windowsize>] `Size of window (in frames) over which to calculate lifetimes/averages. If not specified lifetime/average will be calculated over all frames.`

[name <setname>] `Store results in data sets with name <setname>.`

[averageonly] `Just calculate averages (no lifetime analysis).`

[cumulative] `Calculate cumulative lifetimes/averages over windows.`

[delta] `Calculate difference from previous window average.`

[cut <cutoff>] `Cutoff to use when determining if data is 'present' (default 0.5).`

[greater] `Data is considered present when above the cutoff (default).`

[less] `Data is considered present when below the cutoff.`

[rawcurve] `Do not normalize lifetime curves to 1.0.`

[fuzz <fuzzcut>] `Ignore changes in lifetime state that are less than <fuzzcut> frames.`

[nosort] `Do not sort data sets by name.`

`Data Sets Created:`

<setname> `Number of lifetimes for each set, or if window specified fraction present over time windows.`

<setname>[max] `Maximum lifetime for each set, or if window specified maximum lifetime over time windows.`

<setname>[avg] `Average lifetime for each set, or if window specified average lifetime over time windows.`

<setname>[curve] `Lifetime curves.`

`The following are created only if window not specified:`

<setname>[frames] `Total number of frames lifetime present for each set.`

<setname>[name] `Name of each set.`

Perform lifetime analysis for specified data sets. Lifetime data can either be determined for the entire set, or for time windows of specified size within the set if **window** specified.

A "lifetime" is defined as the length of time something remains 'present'; data is considered present when above or below a certain cutoff (the default is greater than 0.5, useful for analysis of ***hbond*** time series data). For example, in the case of a hydrogen bond **'series'** data set, if a hydrogen bond is present during a frame the value is 1, otherwise it is 0. Given the hbond time series data set {1 1 1 0 1 0 0 0 1 1}, the overall fraction present is 0.6. However, there are 3 lifetimes of lengths 3, 1, and 2 ({1 1 1}, {1}, and {1 1}). The maximum lifetime is 3 and the average lifetime is 2.0, i.e. (3 + 1 + 2) / 3 lifetimes = 2.0. One can also construct a "lifetime curve", which is constructed as the sum of all individual lifetimes. By default these curves are normalized to 1.0, but the raw curve can be obtained using the **rawcurve** keyword. For the example data set here the raw lifetime curve would be 3 frames long:

```
       1 1 1
       1
       1 1
Curve: 3 2 1
```

By default data sets are sorted by name unless **nosort** is specified. The lifetime command can calculate lifetimes over specific time windows by using the **window** keyword. This can be particularly useful if one wants to get a sense for how lifetimes are changing over the course of very long time series data. In addition, averages can be calculated instead of lifetimes by specifying **averageonly**. Cumulative averages over windows can be obtained using the **cumulative** keyword, or the change from the average value in the previous window can be obtained using the **delta** keyword.

The **fuzz** keyword can be used to try and smooth the input data by ignoring changes in state that occur for fewer frames than <fuzzcut>. For example, in the above example hbond time series data set there is a one frame change in state between the first and second lifetimes which could be interpreted as a transient breaking of the hydrogen bond. Using a <fuzzcut> value of 1, this one frame change in state would be ignored, and the data set would effectively appear to lifetime as {1 1 1 1 1 1 0 0 0 1 1}. The state change between the second and third lifetimes is longer than <fuzzcut> (3 frames) and so it would remain.

If **window** is not specified, two files are output: <filename> and crv.<filename>. The file <filename> contains overall lifetime stats for each set with format:

```
#Set <setname> <setname>[max] <setname>[avg] <setname>[frames] <setname>[name]
```

where `<setname>` denotes the total number of lifetimes, `<setname>[max]` denotes the maximum lifetime, `<setname>[avg]` denotes the average lifetime, `<setname>[frames]` denotes the total number of frames present in all lifetimes, and `<setname>[name]` is the data set name. The file crv.<filename> contains the lifetime curves for each set.

If **window** is specified, four files are output: <filename>, max.<filename>, avg.<filename>, and crv.<filename>. <filename> contains the fraction "present" over each time window for each set, max.<filename> contains the maximum lifetime in each time window for each set, avg.<filename> contains the average lifetime over each window for each set, and crv.<filename> contains the overall lifetime curves for each set. For window output, Gnuplot format is recommended.

Example: hbond lifetime analysis

```
parm DPDP.parm7
trajin DPDP.nc
hbond HB out hbond.dat @N,H,C,O series uuseries solutehb.agr \
  avgout hbavg.dat printatomnum
# 'run' is used here to process the trajectory and generate hbond data
run
# Perform lifetime analysis
runanalysis lifetime HB[solutehb] out lifehb.dat
```

Calculate ion lifetimes from hbond over windows of size 100 frames:

```
hbond ION out ion.dat solventdonor :WAT solventacceptor :WAT@O series
run
lifetime HB[solventhb] out ion.lifetime.100.gnu window 100
```

28.10.18. lowestcurve

```
lowestcurve points <# lowest> [step <stepsize>] <dset0> [<dset1> ...]
                   [out <file>] [name <setname>]
```

<# lowest> `Number of lowest points in each bin to average over.`

[step <stepsize>] `Bin step size`

<dset0> [<dset1> ...] `Data set(s) to use.`

[out <file>] `File to write lowest curve to.`

[name <setname>] `Output lowest curve set name.`

Calculate a curve of the average of the # lowest points in bins of stepsize. Essentially each input data set is binned over bins of stepsize, then the lowest <#> points are averaged over for each bin.

28.10.19. meltcurve

```
meltcurve <dset0> [<dset1> ...] [out <outfile>] [name <outsetname>] cut <cut>
```

Calculate melting curve from input data sets (i.e. fraction ’folded’ for each data set) assuming a simple 2-state transition model, using data below <cut>as ’folded’ and data above <cut> as ’unfolded’.

28.10.20. modes

```
modes {fluct|displ|corr|eigenval|trajout|rmsip} name <modesname> [name2 <modesname>]
   [beg <beg>] [end <end>] [bose] [factor <factor>] [calcall]
   [out <outfile>] [setname <name>]
    Options for ’trajout’: (Generate pseudo-trajectory)
   [trajout <name> parm <name> | parmindex <#>
     [trajoutfmt <format>] [trajoutmask <mask>]
    [pcmin <pcmin>] [pcmax <pcmax>] [tmode <mode>]]
    Options for ’corr’: (Calculate dipole correlation)
   { maskp <mask1> <mask2> [...] | mask1 <mask> mask2 <mask> }
   parm <name> | parmindex <#>
Types of Calculations:
```

fluct `RMS fluctuations (X, Y, Z, and total) for each atom across specified normal modes.`

displ `Displacement of cartesian coordinates in the X, Y and Z directions for each atom across specified normal modes.`

corr `Dipole-dipole correlation functions.  Must also specify maskp (see below).`

eigenval `Calculate eigenvalue fractions.`

trajout `Create a pseudo-trajectory along the given mode from the average structure.`

rmsip `Calculate the root-mean-square inner product between modes specified by name and name2.`

Options:

name <modesname> Previously read-in or generated Modes data set name.

[beg <beg>] [end <end>] If modes taken from datafile, beginning and end modes to read. Default for *beg* is 7 (which skips the first 6 zero-frequency modes in the case of a normal mode analysis); for *end* it is 50.

[bose] Use quantum (Bose) statistics in populating the modes.

[factor <factor>] multiplicative constant on the amplitude of displacement/pseudo-trajectory, default 1.0.

[calcall] If specified use all eigenvectors; otherwise eigenvectors associated with zero or negative eigenvalues will be skipped.

[out <outfile>] File to write data results to. If not given results are written to STDOUT.

[setname <name>] Output data set name.

Options for 'trajout':

<name> Output trajectory file name.

[parm <parmfile/tag>|parmindex <#>] Topology file to use (default first Topology loaded).

[trajoutfmt <format>] Output trajectory format.

[trajoutmask <mask>] Mask of atoms that correspond to how modes were originally generated.

[pcmin <pcmin>] Lowest principal component projection value to use for output trajectory.

[pcmax <pcmax>] Highest principal component projection value to use for output trajectory.

[tmode <mode>] Mode to generate pseudo-trajectory for.

Options for 'corr':

[maskp <mask1> <mask2> [...]] If corr, pairs of atom masks (*mask1*, *mask2*; each pair preceded by "maskp" and each mask defining only a single atom) have to be given that specify the atoms for which the correlation functions are desired.

mask1 <mask> mask2 <mask> Instead of maskp, specify two masks; atoms from the first mask will be paired up with atoms from the second mask.

DataSets Created (fluct)

<name>[rmsX] RMS fluctuations in the X direction.

<name>[rmsY] RMS fluctuations in the Y direction.

<name>[rmsZ] RMS fluctuations in the Z direction.

<name>[rms] Total RMS fluctuations.

DataSets Created (displ)

<name>[displX] Displacement in X direction.

<name>[displY] Displacement in Y direction.

<name>[displZ] Displacement in Z direction.

DataSets Created (eigenval)

<name>[Frac] Fraction eigenvalue contributes to overall motion.

<name>[Cumulative] `Cumulative fraction.`

<name>[Eigenval] `Value of eigenvlue.`

`DataSets Created (rmsip)`

<name> `Result of RMSIP calculation.`

Analyze previously calculated eigenmodes obtained from principal component analyses (of covariance matrices) or quasiharmonic analyses (diagmatrix analysis command). Modes are taken from a previously generated data set (i.e. from ***diagmatrix***) or read in from a data file with ***readdata***. By default, classical (Boltzmann) statistics are used in populating the modes. A possible series of commands would be "**matrix** `covar` | `mwcovar` ..." to generate the matrix, "**`diagmatrix`** ..." to calculate the modes, and, finally, "**`modes`** ...".

For example, to calculate the RMS fluctuations or displacements of the first 3 eigenmodes caluclated from a mass-weighted covariance matrix:

```
matrix mwcovar name mwcvmat out mwcvmat.dat
diagmatrix mwcvmat name evecs vecs 5
modes fluct out rmsfluct.dat name evecs beg 1 end 3
modes displ out resdispl.dat name evecs beg 1 end 3
```

Additionally, dipole-dipole correlation functions for modes obtained from principle component analysis or quasiharmonic analysis can be computed.

```
modes corr out cffromvec.dat name evecs beg 1 end 3 \
      maskp @1 @2 maskp @3 @4 maskp @5 @6
```

or

```
mode corr out cffromvec.dat name evecs beg 1 end 3 mask1 @1,3,5 mask2 @2,4,6
```

If **eigenval** is specified, the fraction contribution of each eigenvector to the total motion is calculated and output with format:

```
#Mode Frac. Cumulative Eigenval
```

where `#Mode` is the eigenvector number, `Frac.` is the eigenvalue over the sum of all eigenvalues, `Cumulative` is the cumulative sum of `Frac.`, and `Eigenval` is the eigenvalue itself. Note that in order to get an idea for how much each eigenvector contributes to all motion, this is best used when all possible eigenvectors have been determined for a system.

In order to visualize eigenvectors, pseudo-trajectories along eigenvectors can be created using average coordinates with the **trajout** keyword. For example, to write a pseudo-trajectory of the first principal component from principal component value of -100 to 100 for a previously calculated Modes data set corresponding to heavy atoms (no hydrogens) for residues 1 to 36:

```
parm ../GAAC.nowat.parm7
readdata evecs.dat
runanalysis modes name evecs.dat trajout test.nc trajoutfmt netcdf \
            trajoutmask :1-36&!@H= pcmin -100 pcmax 100 tmode 1
```

28.10.21. multicurve

```
multicurve set <dset> [set <dset> ...]
           <dset> { <equation> |
                    name <dsname> nexp <m> [form {mexp|mexpk|mexpk_penalty} }
           [AX=<value> ...] [out <outfile>] [resultsout <results>]
           [maxit <max iterations>] [tol <tolerance>]
           [outxbins <NX> outxmin <xmin> outxmax <xmax>]
```

set <dset> [set <dset> ...] Data set(s) to fit.

<equation> Equation to fit of form <Variable> = <Equation>. See 28.3.2 on page 559 for more details on equations *cpptraj* understands.

name <dsname> Name of output data sets (required if using nexp).

nexp <m> Fit to specified number of exponentials.

form <type> Fit to specified exponential form:

- **mexp** Multi-exponential, SUM(m)[An * exp(An+1 * X)]
- **mexpk** Multi-exponential plus constant, A0 + SUM(m)[An * exp(An+1 * X)]
- **mexpk_penalty** Same as mexpk except sum of prefactors constrained to 1.0 and exponential constants constrained to < 0.0.

AX=<value> Value of any constants in specified equation with X starting from 0 (can specify more than one).

out <outfile> Write resulting fit curve to <outfile>.

resultsout <results> Write details of the fit to <results> (default STDOUT).

maxit <max iterations> Number of iterations to run curve fitting algotrithm (default 50).

tol <tolerance> Curve-fitting tolerance (default 1E-4).

outxbins <NX> Number of points to use when generating final curve (default same number of points as input data set).

outxmin <xmin> Minimum X value to use for final curve (default same number of points as input data set).

outxmax <xmax> Maximum X value to use for final curve (default same number of points as input data set).

Fit each input data set <dset> to <equation>. See the ***curvefit*** command on page 659 for more details.

28.10.22. multihist

```
multihist [out <filename>] [name <dsname>] [norm | normint] [kde]
          [min <min>] [max <max>] [step <step>] [bins <bins>] [free <T>]
          <dsetarg0> [ <dsetarg1> ... ]
```

out <filename> Output file.

name <dsname> Name for resulting histogram data sets.

norm (Only used if not kde) Normalize so that max bin is 1.0.

normint (Default for kde) Normalize integral over histogram to 1.0.

kde Use kernel density estimator to construct histogram.

min <min> Histogram minimum (default data set minimum).

max <max> Histogram maximum (default data set maximum).

step <step> Histogram step.

bins <bins> Number of histogram bins.

free <T> Calculate free energy from bin populations as G = -R * <T> * ln(Ni / Nmax).

<dsetargX> Data set argument - may specify more than one.

Histogram each data set separately in 1D. Must specify at least **bins** or **step**.

28.10.23. phipsi

```
phipsi <dsarg0> [<dsarg1> ...] resrange <range> [out <file>]
```

<dsargX> `Argument selecting data sets. Can specify more than 1.`

resrange <range> `Residue range to use (actually uses data set index).`

[out <file>] `Output file.`

Calculate the average and standard deviation of [phi] and [psi] data set pairs, write to <file> with format:

```
#Phi Psi SD(Phi) SD(Psi) Legend
```

Where Phi is the average value of phi, Psi is the average value of psi, SD(Phi) is the standard deviation of phi, SD(psi) is the standard deviation of psi, and Legend contains text describing the phi and psi data sets used in the calculation. Periodicity is taken into account during averaging. The data sets must have been internally labeled as type 'phi'/'psi' and must have a data set index set (actions like dihedral and multidihedral do this automatically). For example:

```
parm ../DPDP.parm7
trajin ../DPDP.nc
multidihedral DPDP phi psi
run
phipsi DPDP[phi] DPDP[psi] out phipsi.dat resrange 1-22
```

28.10.24. regress

```
regress <dset0> [<dset1> ...] [name <name>] [out <filename>]
```

dsetX `Data set(s) to perform linear regression for.`

name <name> `Data set name for resulting linear fits.`

out <filename> `File to write fit lines to.`

Perform linear regression on the specified data set(s). Statistics for the fit(s) are reported to STDOUT.

28.10.25. remlog

```
remlog {<remlog dataset> | <remlog filename>} [out <filename>] [crdidx | repidx]
    [stats [statsout <file>] [printtrips] [reptime <file>]] [lifetime <file>]
    [reptimeslope <n> reptimeslopeout <file>] [acceptout <file>] [name <setname>]
```

<remlog dataset> `Previously read-in REM log data.`

<remlog filename> `REM log file name to read in.`

[out <filename>] `Write replica/coordinate index versus time to <filename>.`

- **crdidx** `Print coordinate index vs exchange; output sets contain replica indices.`
- **repidx** `Print replica index vs exchange; output sets contain coordinate indices.`

stats [statsout <file>] `Calculate round-trip statistics and optionally write to <file>.`

printtrips `Print details of each individual round trip.`

[reptime <file>] `Write time spent at each replica to <file>.`

[lifetime <file>] `Print lifetime data at each replica to <file>.`

[reptimeslope <n>] Calculate the slope of time spent at each replica every <n> exchanges.

[reptimeslopeout <file>] File to write reptimeslope output to.

[acceptout <file>] Write overall exchange acceptances to <file>.

[name <setname>] Output data set name.

Analyze previously read in (via ***readdata***) M-REMD/T-REMD/H-REMD replica log data. Statistics calculated include round-trip time, which is the time needed for a coordinate set to travel from the lowest replica to the highest and back, and the number of exchanges each coordinate spent at each replica. For example, to read in REM log data from an Amber M-REMD run and analyze it:

```
readdata rem.log.1.save rem.log.2.save dimfile remd.dim as remlog nosearch
remlog rem.log.1.save stats reptime mremdreptime.dat
```

For an example of ***remlog*** analysis applied to actual REMD data, see Roe et al.[580].

28.10.26. rms2d | 2drms

```
rms2d [crdset <crd set>] [<name>] [<mask>] [out <filename>]
      [dme | nofit | srmsd] [mass]
      [reftraj <traj> [parm <parmname> | parmindex <parm#>] [<refmask>]]
      [corr <corrfilename>]
```

[crdset <crd set>] Name of previously generated COORDS DataSet. If not specified the default COORDS set will be used.

[<mask>] Mask of atoms to calculate 2D-RMSD for. Default is all atoms.

[out <filename>] Write results to <filename>.

[dme] Calculate distance RMSD instead of coordinate RMSD; this is substantially slower.

[nofit] Calculate RMSD without fitting.

[srmsd] Calculate symmetry-corrected RMSD (see 28.9.74 on page 642).

[mass] Mass-weight RMSD.

[reftraj <traj>] Calculate 2D RMSD to frames in trajectory <traj> instead (can also be another COORDS set).

[parm <parmname> | parmindex <#>] Topology to use for <traj>; only useful in conjunction with reftraj.

[<refmask>] Mask of atoms in reference; only useful in conjunction with reftraj.

[corr <corrfilename>] Calculate pseudo-auto-correlation C for 2D-RMSD as $C(i) = \frac{\sum_{j=0}^{j<N-i} \exp(-RMSD(j,j+i))}{N-i}$, where i is the lag, j is the frame #, and N is the total number of frames. An exponential is used to weight the RMSD since 0.0 RMSD is equivalent to correlation of 1.0. This can only be done if reftraj is not used.

DataSet Aspects:

[Corr] (corr only) Pseudo-auto-correlation.

Note: For backwards compatibility with ptraj the command '2drms' will also work.

Calculate the best-fit RMSD of each frame in <crd set> (the default COORDS set if none specified) to each other frame. This creates an upper-triangle matrix named <name> (or a full matrix if **reftraj** specified). The output of the rms2d command can be best-viewed using gnuplot; a gnuplot-formatted file can be produced by giving <filename> a '.gnu' extension. For example, to calculate the RMSD of non-hydrogen atoms of each frame in trajectory “test.nc” to each other frame, writing to a gnuplot-viewable file “test.2drms.gnu”:

```
trajin test.nc
rms2d !(@H=) rmsout test.2drms.gnu
```

To calculate the RMSD of atoms named CA of each frame in trajectory "test.nc" to each frame in "ref.nc" (assuming test.nc and ref.nc are using the default topology file):

```
trajin test.nc
rms2d @CA rmsout test.2drms.gnu reftraj ref.nc
```

28.10.27. rmsavgcorr

```
rmsavgcorr [crdset <crd set>] [<name>] [<mask>] [out <filename>] [mass]
        [stop <maxwindow>] [offset <offset>]
        {reference <ref file> parm <parmfile> | first}
```

[crdset <crd set>] `COORDS data set to use (if not specified the default COORDS set will be used).`

[<name>] `Output data set name.`

[<mask>] `Atoms to calculate RMS average correlation for.`

[out <filename>] `Output filename.`

[mass] `Mass weight the RMSD calculation.`

[stop <maxwindow>] `Only calculate RMS average correlation up to <maxwindow>.`

[offset <offset>] `Skip every <offset> windows in calculation.`

[first] `Use first averaged frame as reference for each window (default).`

[reference <ref file> [parm <parmfile>] `Use reference file (with specified parm) as reference for each window.`

The RMS average correlation[556] (RAC) is calculated as the average RMSD of running-averaged coordinates over increasing window sizes (or lag). Output has format:

```
<WindowSize> <RAC>
```

The first entry has a window size of 1, and so is just the average RMSD of all frames to the specified reference structure. The second entry has a window size of two, so it is the average RMSD of all frames averaged over two adjacent windows to the specified reference, and so on. The RAC will be calculated up to the number of frames minus 1 or the value specified by **stop**, whichever is lower. The offset can be used to speed up the calculation by skipping window sizes. To calculate mass-weighted RMSD specify **mass**. Note that to reduce memory costs it can be useful to strip all coordinates not involved in the RMS fit from the system prior to specifying 'rmsavgcorr'. For example, to calculate the correlation of C-alpha RMSD of residues 2 to 12:

```
strip !(:2-12@CA)
rmsavgcorr out rmscorr.dat
```

The curve generated by RAC decays towards zero due to the way RAC is defined. By the time the "lag" is N-1 (where N is the total number of frames) you have only two averaged coordinates: call them Avg1 (averaged over 1 though N-1 frames) and Avg2 (averaged over 2 through N frames). Barring any extraordinary circumstances the RMSD between Avg1 and Avg2 will almost certainly be quite low.

The RAC is a way to probe the time scales of interesting events. Any deviation from a smoothly decaying curve is an indication that there are some significant structural differences occurring over that time interval. RAC curves can be particularly useful when comparing independent simulations of the same system.

One thing to keep in mind that since the underlying metric is RMSD, it can be sensitive to the reference frame you choose. It may be useful to try looking at both RAC from the first frame, as well as an averaged reference frame. For an example of use see Galindo-Murillo et al.[581], in particular Figure 2.

28.10.28. rotdif

```
rotdif [outfile <outfilename>] [usefft]
  Options for generating random vectors:
[nvecs <nvecs>] [rvecin <randvecIn>] [rseed <random seed>]
[rvecout <randvecOut>] [rmatrix <set name> [rmout <rmOut>]]
  Options for calculating vector time correlation functions:
[order <olegendre>] [ncorr <ncorr>] [corrout <corrOut>]
  *** The options below only apply if 'usefft' IS NOT specified. ***
  Options for calculating local effective D, small anisotropy:
[deffout <deffOut>] [itmax <itmax>] [tol <tolerance>] [d0 <d0>]
[nmesh <NmeshPoints>] dt <tfac> [ti <ti>] tf <tf>
  Options for calculating D with full anisotropy:
[amoeba_tol <tolerance>] [amoeba_itmax <iterations>]
[amoeba_nsearch <n>] [scalesimplex <scale>] [gridsearch]
  *** The options below only apply if 'usefft' IS specified. ***
  Options for curve-fitting:
[fit_tol <tolerance>] [fit_itmax <max # iterations>]
```

outfile <outfilename> File to write all output from rotdif command to.

Options for generating random vectors:

nvecs <nvecs> Number of random vectors to generate (default 1000).

rvecin <randvecIn> File to read random vectors from (format is 1 per line, 4 columns, <#> <VX> <VY> <VZ>).

rseed <random seed> Seed for random number generator (default 80531). Specify -1 to use wallclock time.

rvecout <randvecOut> File to write random vectors to (format is 1 per line, 4 columns, <#> <VX> <VY> <VZ>).

rmatrix <set name> Data set to read rotation matrices from. Rotation matrices will be used to rotate random vectors.

rmout <rmOut> Write rotation matrices to file, 1 per line, frame # followed by matrix in row-major order.

Options for calculating vector time correlation functions:

order <olegendre> The order of Legendre polynomials to use when calculating vector time correlation functions (default 2).

ncorr <ncorr> Maximum length of time correlation functions in frames. If this is not specified it will be set to (tf - ti) / dt (recommended).

corrout <corrOut> If specified write vector time correlation functions to <corrOut>.X with format: <Time> <Px>

Options for calculating local effective D, small anisotropy:

deffout <deffOut> File to write out local effective diffusion constants determined in the limit of small anisotropy.

itmax <itmax> Maximum number of iterations to determine each local effective diffusion constant (small anisotropy) assuming fit to single exponential form (default 500).

tol <tolerance> Tolerance for determining local effective diffusion constant (small anisotropy) assuming fit to single exponential form (default 1E-6).

d0 <d0> `Initial guess for small anisotropy diffusion constant in radians^2/ns (default 0.03).`

nmesh <NmeshPoints> `Number of points per frame to use when creating cubic-splined-smoothed forms of vector time correlation curves (default 2).`

dt <tfac> `Time interval between frames (used in integrating vector time correlation curves) in ns.`

ti <ti> `Initial time value in ns for integrating the time correlation functions (default 0.0).`

tf <tf> `Final time value in ns for integrating the time correlation functions. It is recommended this be less than the maximum simulation time since the tails of time correlation functions tend to be noisy.`

Options for calculating D with full anisotropy:

amoeba_tol <tolerance> `Tolerance for downhill-simplex minimizer (default 1E-7).`

amoeba_itmax <iterations> `Number of iterations to run downhill-simplex minimizer (default 10000).`

amoeba_nsearch <n> `Number of searches to perform with downhill-simplex minimizer (default 1).`

scalesimplex <scale> `Factor to use when scaling simplexes (default 0.5).`

gridsearch `If specified, perform a brute-force grid search to attempt to find a better solution for diffusion tensor with full anisotropy (may be expensive).`

Evaluate rotational diffusion properties of a molecule over a trajectory according to an expanded version of the procedure laid out by Wong & Case[582]. Briefly, random vectors (representing the orientation of the molecule) are rotated according to rotation matrices obtained from an RMS fit to a reference structure (typically an averaged structure). For each random vector the time correlation function of the rotated vector is calculated using Legendre polynomials of the specified order. The integral over this time correlation function (which may be smoothed using cubic splines to improve the integration) is then used to find the effective diffusion constant (D) in the limit of small anisotropy. Then, using each calculated D, the diffusion tensor is determined with full anisotropy. Finally, a downhill simplex minimizer is used to optimize D with full anisotropy; (this last step is not described in the original paper).

Rotation matrices are generated via an RMS fit to a reference structure (see 28.9.60 on page 635). It is recommended that the RMS fit be done to an average structure (see 28.9.7 on page 593). These rotation matrices are used to rotate each random vector M times (where M is the total number of frames), which creates a time series for each random vector. The time correlation functions are calculated for each random vector time series using Legendre polynomials of the specified order (default 2). The maximum length of the correlation function (or lag) can be specified by **ncorr** (in frames). If **ncorr** is not specified it will be set internally based on the specified values of **ti**, **tf**, and **dt**; this is recommended. Note that if **ncorr** is specified it should be set to a number less than the total number of frames since noise in time correlation functions increases as **ncorr** approaches the # of frames. The integration over the correlation function is from **ti** (in whatever units are used of **dt**, generally ns; 0.0 ns if not specified) to **tf** (same units as **ti**), with the time between frames specified by **dt**; the final time should be less than the total simulation time (see example below).

The relative size of the mesh used with cubic spline interpolation for integration is controlled by **nmesh** (size of the mesh is **ncorr** points * **nmesh**); **nmesh** = 1 means no interpolation, default is 2. The iterative solver for effective value of the diffusion constant from the correlation functions is controlled by **itmax**, **tol**, and **d0**, where **itmax** specifies the number of iterations to perform (default 500), **tol** specifies the tolerance (default 1E-6), and **d0** specifies the initial guess for the diffusion constant in radians^2 / ns (default 0.03). Effective diffusion constants for each random vector can be written out to a file specified by **deffout**. Results are printed to the file specified by **outfile**. Details on the Q and D tensors are given, as well as observed and calculated tau for each random vector. First, results are printed for analysis in the limit of small anisotropy. Next, results are printed for analysis with full anisotropy. The results of the full anisotropic calculation are first given using results from the small anisotropic analysis as an initial guess, followed by the final results after minimization using the downhill simplex (amoeba) minimizer.

Example

There are two important things to keep in mind when using rotdif analysis:

1. When calculating any kind of diffusive property it is best to simulate in the microcanonical (NVE) ensemble with a shorter time step and increased SHAKE tolerance; thermostats and barostats will effect diffusion calculations.
2. Time correlation functions become noisier as the length of the function approaches the maximum. Therefore in general one should choose parameters for the time correlation function that are much shorter than the total simulation length.

For example, given a trajectory 'mdcrd.nc' containing 10000 frames with a total simulation time of 200 ns (so the time between frames is 0.02 ns), to calculate rotational diffusion using 100 vectors using rotation matrices generated via an RMS fit to 'avgstruct.pdb', computing and integrating the time correlation function for each vector from 0 to 5 ns (1/40th of the simulation), and writing out the effective diffusion constants and results to 'deffs.dat' and 'rotdif.out' respectively:

```
reference avgstruct.pdb [avg]
rms R0 @CA,C,N,O ref [avg] savematrices
trajin mdcrd.nc
rotdif nvecs 100 rmatrix R0[RM] \
       ti 0.0 tf 5.0 dt 0.02 deffout deffs.dat \
       outfile rotdif.out
```

28.10.29. runningavg

```
runningavg <dset1> [<dset2> ...] [name <dsetname>] [out <filename>]
                   [ [cumulative] | [window <window>] ]
```

<dset1> [<dset2> ...] `Data set(s) to calculate running average for.`

[name <dsetname>] `Output running average data set name.`

[out <filename>] `File to write results to.`

[cumulative] `Calculate cumulative running average instead.`

[window <window>] `Size in frames of window over which to calculate running average.`

Calculate running average over windows of given size for data in selected data set(s).

28.10.30. spline

```
spline <dset0> [<dset1> ...] [out <outfile>] [meshsize <n> | meshfactor <x>]
[meshmin <mmin>] [meshmax <mmax>]
```

<dsetX> `Data set(s) to perform splining on.`

[out <outfile>] `Write splined data to <outfile>.`

[meshsize<n>] `Size of the mesh to use for splining.`

[meshfactor <x>] `If meshsize is not given, use a mesh of data set size * <x>.`

[meshmin <mmin>] `Mesh X minimum value.`

[meshmax <mmax>] `Mesh X maximum value.`

Cubic spline the given data sets.

28.10.31. statistics | stat

```
stat {<name> | ALL} [shift <value>] [out <filename>] [noeout <filename>]
 [ignorenv] [name <noe setname>]
```

<name> `Name of data set to analyze.`

ALL `analyze all data sets.`

shift <value> `Subtract <value> from all elements in each data set.`

[out <filename>] `Write analysis results to <filename> (STDOUT if not specified).`

[noeout <filename>] `(Type 'noe' only) Write summary of NOE results to <filename>.`

[ignorenv] `(Type 'noe' only) Ignore negative NOE violations (i.e. shorter-than-expected distances).`

[name <noe setname>] `(Type 'noe' only) Name for output NOE data sets.`

`DataSet Aspects for type 'noe' output:`

[R6] `Averaged` $1/r^6$`distance for each set.`

[NViolations] `Number of violations based on given bounds for each set.`

[AvgViolation] $1/r^6$ `averaged distance minus expected distance for each set.`

[NOEnames] `Name of each set.`

Analyze angles, dihedrals, distances, and/or puckers and calculate various properties. More specific analyses can be obtained by labelling distances/dihedrals/puckers (from e.g. the ***distance***, ***dihedral***, ***pucker*** commands or with the ***dataset*** command) with the 'type <label>' keyword:

dihedral type labels: alpha, beta, gamma, delta, epsilon, zeta, chi, c2p h1p, phi, psi, omega, pchi

distance type labels: noe

pucker type labels: pucker

For each input data set, the average, standard deviation, initial and final values will be reported. The cyclic nature of dihedral/pucker data sets is taken into consideration when averaging.

28.10.31.1. Torsion Analysis

A table will be written in ASCII format showing the distribution of torsion values for each data set. More specific information may be printed based on the set type. Values in the output marked SNB are from those defined by Schneider, Neidle, and Berman.[583] For more information on nucleic acid torsion as pertains to RNA see further work by Schneider et al..[584]

For example, to perform in-depth analysis on some nucleic acid dihedral angles:

```
dihedral g0 out dihedrals.dat :1@O5' :1@C5' :1@C4' :1@C3' type gamma
dihedral d0 out dihedrals.dat :1@C5' :1@C4' :1@C3' :1@O3' type delta
dihedral c0 out dihedrals.dat :1@O4' :1@C1' :1@N9 :1@C4 type chi
analyze statistics all out stat.dat
```

28.10.31.2. Distance Analysis

A table will be written in ASCII format showing the distribution of distance values < 6.5. If a distance is labled as 'type noe' a compact time series will be printed in ASCII format showing the NOE as strong, medium, or weak. In addition the <r^-6>^(-1/6) averaged value will be reported, as well as the number of upper/lower bound violations. If **'noeout'** is specified, a summary of these results will be written with format:

```
<#NOE> <R6> <Nviolation> <AvgViolation> <Name>
```

Where <#NOE> is an index, <R6> is the <r^-6>^(-1/6) averaged distance, <Nviolation> is the total number of bounds violations, <AvgViolation> is the average difference from expected distance *Rexp* when the distance is violated (note that if not explicitly set, *Rexp* is set to the upper bound when the lower bound is 0.0, or the average of upper and lower bounds otherwise), and <Name> is the data set legend.

For example, the following input could be used to check certain distances for NOE violations:

```
distance :3@HB= :10@HG= type noe noe_medium
distance :3@HE= :10@HG= type noe noe_strong
distance :3@HA :12@HA type noe noe_medium
distance :3@HD= :12@HG= type noe noe_medium
distance :3@HE= :12@HA type noe noe_strong
analyze statistics all out dpdp.noe.dat noeout noe_graph.dat name Res3_NOE
```

28.10.31.3. Pucker Analysis

A table will be written in ASCII format showing the distribution of pucker phases for each data set.

28.10.32. timecorr

```
timecorr vec1 <vecname1> [vec2 <vecname2>] out <filename>
           [order <order>] [tstep <tstep>] [tcorr <tcorr>]
           [dplr] [norm] [drct] [dplrout <dplrfile>] [ptrajformat]
```

vec1 <vecname1> [vec2 <vecname2>] Vector(s) on which to operate. By default the auto-correlation function will be calculated if one vector is specified, and the cross-correlation function will be calculated if two vectors are specified.

out <filename> Name of file to write output to.

[order <order>] Order of Legendre polynomials to use; default 2.

[tstep <tstep>] Time between snapshots (default 1.0).

[tcorr <tcorr>] Maximum time to calculate correlation functions for (default 10000.0).

[dplr] Output correlation functions $C_l \equiv < P_l/(r(0)^3 r(\tau)^3) >$ and $< 1/(r(0)^3 r(\tau)^3) >$ in addition to the P_l correlation function.

[norm] Normalize all correlation functions, i.e., $C_l(t=0) = P_l(t=0) = 1.0$.

[drct] Use the direct method to calculate correlations instead of FFT; this will be much slower.

[dplrout] (dplr only) Write extra information for each vector related to dplr option to <dplrfile>.

[ptrajformat] Write output in ptraj style (prevents use of data formatting options).

DataSet Aspects:

[P] P<order> correlation function.

[C] C<order> correlation function (dplr only).

[R3R3] <1/(r(0)^{3}r(t)^3> correlation function (dplr only).

[R] (_TC_DIPOLAR_) Average magnitude (<R>).

[RRIG] (_TC_DIPOLAR_) Sqrt(<R^2>).

[R3] (_TC_DIPOLAR_) <1/R^3>.

[R6] (_TC_DIPOLAR_) <1/R^6>.

[Name] `(_TC_DIPOLAR_) Vector name.`

Calculate time auto/cross-correlation functions for vectors using spherical harmonics theory. NOTE: To calculate direct correlation functions for vectors just use the ***corr*** analysis command. The **norm** keyword will normalize the resulting correlation functions. Note that if **dplr** is specified, a new global data set named _TC_DIPOLAR_ will be created, containing extra data for each vector analyzed with a '***timecorr*** **dplr**' command.

Examples

Vectors between atoms 5 and 6 as well as 7 and 8 are calculated below, for which auto and cross time correlation functions are obtained.

```
vector v0 @5 @6
vector v1 @7 @8
timecorr vec1 v0 tstep 1.0 tcorr 100.0 out v0.out order 2
timecorr vec1 v1 tstep 1.0 tcorr 100.0 out v1.out order 2
timecorr vec1 v0 vec2 v1 tstep 1.0 tcorr 100.0 out v0_v1.out order 2
```

Similarly, a vector perpendicular to the plane through atoms 18, 19, and 20 is obtained and further analyzed.

```
vector v2 @18,@19,@20 corrplane
timecorr vec1 v3 tstep 1.0 tcorr 100.0 out v2.out order 2
```

28.10.33. vectormath

```
vectormath vec1 <vecname1> vec2 <vecname2> [out <filename>] [norm] [name <setname>]
           [ dotproduct | dotangle | crossproduct ]
```

vec1 <vecname1> vec2 <vecname2> `Vector(s) on which to operate.`

[out <filename>] `Name of file to write output to.`

[dotproduct] `(Default) Calculate the dot-product of the two vectors.`

[dotangle] `Calculate angle from dot-product between the two vectors; vectors will be normalized.`

[crossproduct] `Calculate cross-product of the two vectors.`

[norm] `Normalize the vectors; this will affect any subsequent calculations with the vectors.  This is turned on automatically if dotangle specified.`

Calculate dot product, angle from dot product (degrees), or cross product for specified vectors. Note that **norm** normalizes the vectors themselves; the vectors will remain normalized for subsequent calculations or output. Either **vec1** or **vec2** can be of size 1; in that case each vector in the set with N frames operates on the single vector. For example, if **vec1** is size N and **vec2** is size 1, then each frame of vec1 is operated on the single vector from vec2.

For example, to get the angles between two previously calculated vectors v1 and v2:

```
vectormath vec1 v1 vec2 v2 dotangle out dotproduct.dat name acos(|V1|*|V2|)
```

28.10.34. wavelet

```
wavelet [crdset <set name>] nb <n scaling vals> [s0 <s0>] [ds <ds>]
        [correction <correction>] [chival <chival>] [type <wavelet>]
        [out <filename>] [name <setname>]
        [cluster [minpoints <#>] [epsilon <value>] [clusterout <file>]
           [clustermapout <file>] [cmapdetail] [kdist] [cprefix <PDB prefix>]
           [overlay <trajfile>] [overlayparm <parmfile>]]
```

[crdset <set name>] `COORDS data set to use`

nb <n scaling vals> `Number of scales. The smaller the number the better resolution, but slower to plot.`

[s0 <s0>] `The smallest scale of the wavelet function (default 2dt where dt is time between snapshots in ps )`

[ds <ds>] `Spacing between discrete scales. (Default is 0.25. Smaller value of ds gives finer resolution. The largest values that give adequate sampling in scale for Morlet and Paul are 0.5 and 1.5, respectively)`

[correction <correction>] `The scale-to-wavelength parameter (1.01 for Morlet, 1.389 for Paul). Automatically set based on wavelet if not otherwise specified.`

[chival <chival>] `The value of` χ_2^2`at a particular confidence level`

[type <wavelet>] `Type of wavelet function to use <morlet> or <paul>`

[out <filename>] `Write results to file named <filename>`

[name <setname>] `Store results in data set with name <setname>`

[cluster] `Perform wavelet clustering i.e. wavelet feature extraction analysis.`

- **[minpoints <#>]** `Minimum number of points necessary to form a region of interest.`
- **[epsilon <value>]** `Minimum region of interest size.`
- **[clusterout <file>]** `Output for clustering (see below).`
- **[clustermapout <file>]** `Output cluster map (recommended gnuplot format, see below).`
- **[cmapdetail]** `Instead of the map being smoothed to cluster regions, show full detail.`
- **[kdist]** `Can be used to determine minpoints and epsilon - see below.`
- **[cprefix <PDB prefix>]** `Output cluster region PDBs (only containing from minimum to maximum atom and minimum to maximum frame) with given prefix.`
- **[overlay <trajfile>]** `Create a trajectory that can be overlaid with the original trajectory to highlight atoms of interest. Atoms in cluster regions will get their normal coordinates - all others are set to the common center of mass.`
- **[overlayparm <parmfile>]** `Topology that can be used with the overlay trajectory.`

`<wavelet>: morlet, paul`

Perform the wavelet analysis using fast Fourier transform (FFT) algorithm on specified trajectory and write out to a gnuplot-formatted file named <name.gnu>. The created Wavelet map provides a clear picture of the significant motions which are characterized both in time and space. Note that typically the trajectory in question should have rotational and translational movement removed (via e.g. the ***rms*** command); otherwise these will be reflected in the wavelet analysis results.

Wavelet analysis contains two main steps which performs continues wavelet transform (CWT) and statistical significance testing as proposed by Torrence and Compo[585]. Analysis is executed on one dimensional (1-D) coordinate which is defined as the displacement from the starting position. For each atom, CWT is calculated over a specified range of scales from S_0up to $S_0 2^{(nb-1)ds}$. To obtain the CWT of the trajectory the Fourier transform of atom's displacement and wavelets which scaled by S (S is calculated from: $S = S_0 2^{jds}$; $j = 0, 1, 2, \ldots, nb-1$) is computed and then the inverse Fourier transform of the product of Fourier transforms will be calculated as the CWT. After calculating the wavelet coordinates for all atoms, a significance testing is performed to determine the significance of each wavelet coordinate. For doing this test we need to have an appropriate background spectrum to consider as a mean or expected spectrum and compare our wavelet coordinates against this background. In order to calculate the background spectrum since wavelet spectrum (according to the convolution theorem) follows the Fourier spectrum, the Fourier coefficients over every atom's displacement is calculated using the following formula and a model (μ_k) is constructed on average which Fourier coefficients fit (X_n) is the time series which is the atom's displacement and k is the frequency index[586].

$$f_{k=\frac{1}{N}} \sum_{n=0}^{N-1} \exp\left(\frac{-2\pi ikn}{N}\right) X_n$$

This test is implemented based on the null hypothesis that the assumption is that Fourier coordinates normally distributed around the expected value, then the wavelet coordinates should also be normally distributed. Assuming the expected background spectrum and since the square of a normally distributed variable is chi-square distributed, then the distribution for the square of the absolute values of wavelet coordinates ($|W_{i,k}|^2$ is as follows ($\sigma^2$is the variance of the atom's displacement).

$$\sigma^2 \mu_k \chi_2^2/2$$

Then choosing a confidence level we can determine the minimum acceptable value for $|W_{i,k}|^2$to be considered as a significant coordinates at that certain confidence level. In the final map the scales of only those wavelet coordinates which are significantly above the expected distribution are stored.

For example, to perform wavelet analysis on residues 1 to 17 with 40 scaling values starting from scaling of 0.2 with a spacing of 0.25 using the Morlet wavelet:

```
parm nowat.withions.parm7
trajin nowat.image.nc
rms :1-17@C*,N*,O*,P* first mass
wavelet nb 40 s0 0.2 ds 0.25 correction 1.01 chival 1.6094 type morlet \
        :1-17 out wavelet.gnu usemap
```

Wavelet Analysis Feature Extraction

Wavelet analysis feature extraction (WAFEX)[587] uses a density-based clustering algorithm (a modified version of the DBSCAN algorithm) to highlight physical and temporal regions that have significant motions from wavelet mapsand can extract the specific atoms and frames involved in these motions for further analysis. Cluster regions shown in the map will be smoother by default for easier visualization (unless **cmapdetail** is specified). Details of the clustering are provided via the **clusterout** keyword with format:

```
#Cluster [points] [minatm] [maxatm] [minfrm] [maxfrm] [avgval]
```

#Cluster `Cluster region number.`

points `Number of points in the cluster.`

minatm `Starting atom of the region.`

maxatm `End atom of the region.`

minfrm `Starting frame of the region.`

maxfrm `End frame of the region.`

avgval `Average value of points in the region.`

For example, to create a 2D gnuplot map highlight regions of interest called 'cluster.gnu' one could use the following input.

```
parm ../DPDP.parm7
trajin ../DPDP.nc
rms @C,CA,N first
wavelet nb 10 s0 2 ds 0.25 type morlet correction 1.01 chival 0.25 \
        :1-22 name DPDP \
          cluster clustermapout cluster.gnu clusterout cluster.dat \
            minpoints 66 epsilon 10.0
datafile cluster.gnu usemap palette kbvyw
```

Some experimentation with **kdist** may be required to obtain reasonable values for **minpoints** and **epsilon**. See 28.10.4 on page 657 as well as the Heidari et al paper for further discussion.

28.11. Analysis Examples

Please note that typically for principal component analysis (PCA) the trajectory needs to be aligned against a reference structure to remove overall global and translation motion. Use the ***rms*** command for this.

28.11.1. Cartesian covariance matrix calculation and projection (PCA)

After calculating modes, snapshots can be projected onto these in an additional pass through the trajectory. It is very important that the snapshots used when projecting are exactly the same as those used to generate the original covariance matrix. This example takes advantage of the COORDS data set functionality in cpptraj to save snapshots for the purposes of projection.

```
# Step one. Generate average structure.
# RMS-Fit to first frame to remove global translation/rotation.
parm myparm.parm7
trajin mytraj.nc
rms first !@H=
average crdset AVG
run
# Step two. RMS-Fit to average structure. Calculate covariance matrix.
# Save the fit coordinates.
rms ref AVG !@H=
matrix covar name MyMatrix !@H=
createcrd CRD1
run
# Step three. Diagonalize matrix.
runanalysis diagmatrix MyMatrix vecs 2 name MyEvecs
# Step four. Project saved fit coordinates along eigenvectors 1 and 2
crdaction CRD1 projection evecs MyEvecs !@H= out project.dat beg 1 end 2
```

28.11.2. Dihedral covariance matrix calculation and projection for backbone phi/psi (PCA)

```
parm ../1rrb_vac.prmtop
trajin ../1rrb_vac.mdcrd
# Generation of phi/psi dihedral data
multidihedral BB phi psi resrange 2
run
# Calculate dihedral covariance matrix and obtain eigenvectors
matrix dihcovar dihedrals BB[*] out dihcovar.dat name DIH
diagmatrix DIH vecs 4 out modes.dihcovar.dat name DIHMODES
run
# Project along eigenvectors
projection evecs DIHMODES out dih.project.dat beg 1 end 4 dihedrals BB[*]
run
```

29. pytraj

29.1. Introduction

pytraj [588] is Python front end of cpptraj. It is written to introduce more flexibility in data analysis by combining with Python's rich ecosystems (such as *numpy*, *scipy*, *pandas*, *scikit-learn*, *ipython-notebook*, etc.). It is aimed at users who are familiar with Python and want to combine cpptraj's functionality with the flexibility of Python. It is still very new, and in active development, and users should be aware that some of the syntax (i.e., the *API*) may change in future versions.

This project is not intended to replace cpptraj, but rather to extend its functionality by placing allowing seamless and efficient data interchange between cpptraj and Python. Therefore, this project is aimed at users who are either comfortable and familiar with the Python programming language or wish to become so. You should be familiar with basic programming concepts (like conditionals, loops, and arrays) and preferably Python syntax before trying to use pytraj.

29.2. Development

If you are interested in contributing to the development of pytraj, or you want to build the source code directly, either fork or clone the repository from Github at `https://github.com/amber-md/pytraj`. Note that this method of installation is more complex, as you will need to obtain and build an updated version of libcpptraj (instructions can be found in the pytraj Github project). Also, you can check developer guide `http://amber-md.github.io/pytraj/latest/developer_guide.html`.

29.3. Documentation and examples

Useful links are listed below.

- The pytraj Github repository: `https://github.com/amber-md/pytraj`
- Comprehensive documents and tutorials can be found in `https://amber-md.github.io/pytraj`
- Example scripts: `https://github.com/amber-md/pytraj/tree/master/examples`
- We highly suggest user to use Jupyter notebook for interactive computing: `http://jupyter.org/`

29.3.1. Minimal examples

Only several highlight features of pytraj are shown here.

29.3.1.1. Loading trajectories to memory

```
import pytraj as pt

# load all frames into memory if filesize is small
traj = pt.load('tz2.nc', top='tz2.parm7')

# load but skip every 10 frames
traj = pt.load('tz2.nc', top='tz2.parm7', stride=10)
```

```
# load specific frame numbers
traj = pt.load('tz2.nc', top='tz2.parm7', frame_indices=[0, 8, 10, 20])

# load with given mask
traj = pt.load('tz2.nc', top='tz2.parm7', mask='@CA')
```

29.3.1.2. Lazy loading trajectories

Sometimes the trajectories are to large to load to memory, users can use pytraj.iterload method.

```
import pytraj as pt

traj = pt.iterload('tz2.nc', 'tz2.parm7')

# load several files
traj = pt.iterload('tz2.*.nc', 'tz2.parm7')

# load several files by explicitly giving filenames
traj = pt.iterload(['tz2.0.nc', 'tz2.1'.nc'], 'tz2.parm7')
```

Please check `http://amber-md.github.io/pytraj/latest/trajectory_exercise.html` for further information.

29.3.1.3. Perform calculation

```
# radgyr
data = pt.radgyr(traj, mask='@CA')

# rmsd to 1st frame, mask='@CA'
data = pt.rmsd(traj, ref=0, mask='@CA')

# rmsd to specific reference
ref = traj[3]
data = pt.rmsd(traj, ref=ref, mask='@CA')

# compute distance
pt.distance(traj, ':1-3 :5-8')

# load pdb from RCSB website then perform hbond calculation
traj = pt.fetch_pdb('1l2y')
hbond_data = pt.hbond(traj)
print(hbond_data.donor_aceptor)
```

More analysis commands can be found in `http://amber-md.github.io/pytraj/latest/_api/pytraj.all_actions.html`.

29.3.1.4. Writing trajectory

```
import pytraj as pt

# write whole trajectory
pt.write_traj('output.nc', traj, overwrite=True)

# write specific frames
```

```
pt.write_traj('output.nc', traj, frame_indices=[0, 8, 9], overwrite=True)

# user can use save method
traj.save('output.nc')

# if you are using Ipython/Jupyter notebook (or interactive terminal), you can get help by
pt.write_traj?
# the hit <Enter>

# converting netcdf format to DCD
traj = pt.iterload('tz2.nc', 'tz2.parm7')
traj.save('tz2.dcd')
```

29.3.1.5. Combine pytraj with pysander for energy evaluation

```
import pytraj as pt
traj = pt.iterload('tz2.nc', 'tz2.parm7')

# compute energies (potential, bond, angle, dihedral, GB, ...), use igb=8 model
energy_dict = pt.energy_decomposition(traj, igb=8)

# get energy for different component
energy_dict['tot']
energy_dict['gb']
```

29.3.1.6. Parallel calculation

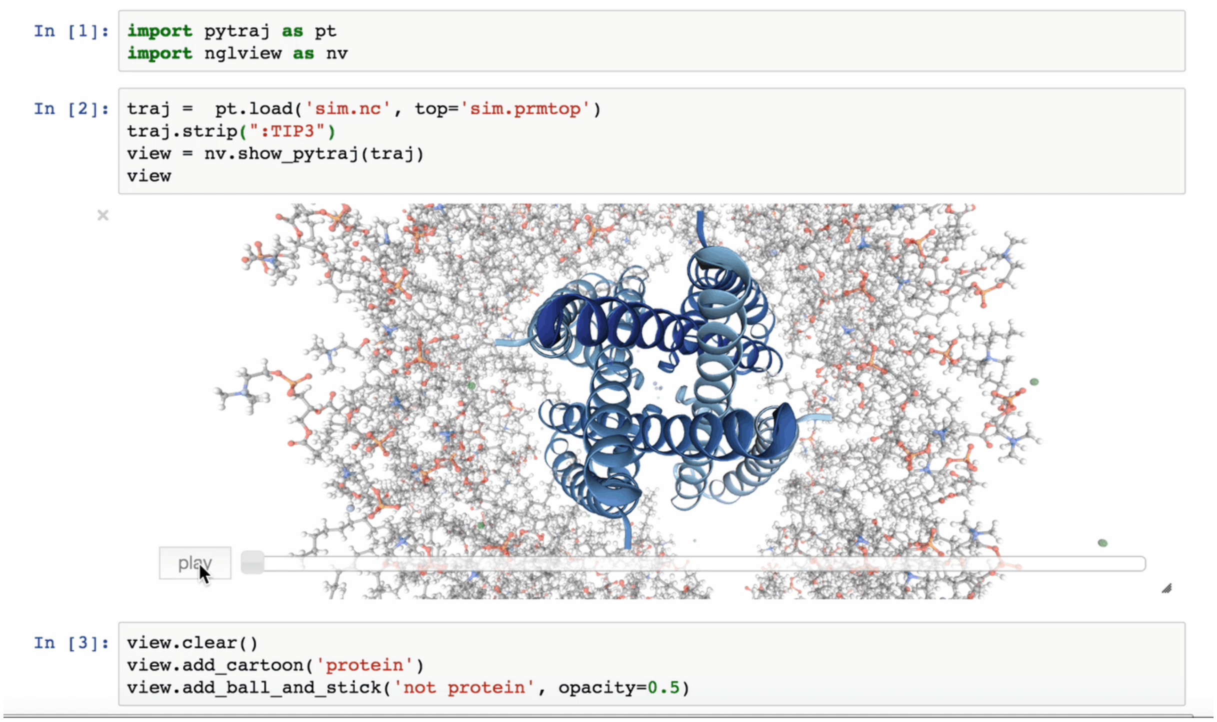
pytraj supports parallel calculation through multiprocessing (python) or MPI (require mpi4py)

```
import pytraj as pt

# for parallel calculation, pytraj only supports `iterload` method.
traj = pt.iterload('tz2.nc', top='tz2.parm7')

# serial version
data = pt.radgyr(traj, '@CA')

# multiprocessing
data = pt.pmap(pt.radgyr, traj, '@CA', n_cores=6)
energy_dict = pt.pmap(pt.energy_decomposition, traj, igb=8, n_cores=6)

# user can chain a series of cpptraj commands for parallel calculation too
data = pt.pmap(['rms', 'radgyr @CA nomax', 'surf @CA'], traj, n_cores=8)

# mpi
data = pt.pmap_mpi(pt.radgyr, traj, '@CA')
```

Please check: `http://amber-md.github.io/pytraj/latest/tutorials/tutorial_parallel.html`

29.3.1.7. Simplify Principal Component Analysis (PCA) caculation

```
traj = pt.load('tz2.nc', 'tz2.parm7')
data = pt.pca(traj, mask='@CA', n_vecs=2)
# get document for this method
print(pt.pca.__doc__)
```



```
In [1]: import pytraj as pt
        import nglview as nv

In [2]: traj =  pt.load('sim.nc', top='sim.prmtop')
        traj.strip(":TIP3")
        view = nv.show_pytraj(traj)
        view

In [3]: view.clear()
        view.add_cartoon('protein')
        view.add_ball_and_stick('not protein', opacity=0.5)
```

Figure 29.1.: *Example of trajectory viewer in Jupyter notebook*

Please also check: `http://amber-md.github.io/pytraj/latest/tutorials/tut_pca.html`.

29.3.1.8. Fancy indexing of trajectory

```
# get new Trajectory by skipping every 2 frames
traj[::2]

# get new Trajectory but only keeping coordinates of CA atoms
traj['@CA']

# get new Trajectory with given frame numbers
framelist =[0, 3, 7]
traj[framelist]
```

Please see also: `http://amber-md.github.io/pytraj/latest/trajectory_slice.html`

29.3.1.9. Trajectory Viewer

Trajectory can be quickly viewed in Jupyter notebook by using pytraj and nglview (`https://github.com/arose/nglview`).User can install nglview via pip by “amber.pip install nglview” or via conda by “conda install nglview -c bioconda”

30. MMPBSA.py

Note: Amber now has three(!) scripts to carry out MM-PBSA-like calculations. The one described here (the "python" version) is more recent, generally simpler to use, and has a more active support community for answering questions. The *amberlite* code (described in Chapter 41) is more limited, and focused on protein-ligand interactions; it is a great place for users new to AmberTools to begin. The version described in Chapter 31 (the "perl" version) continues to be updated, and has some specialized features. Most new users should try the python or amberlite versions first.

None of these should be considered as a "black-box", and users should be familiar with Amber before attempting these sorts of calculations. These scripts automate a series of calculations, and cannot trap all the types of errors that might occur. ***You should be sure that you know how to carry out an MM-PBSA calculation "by hand" (i.e., without using the scripts);*** if you don't understand in detail what is going on, you will have no good reason to trust the results. Also, if something goes awry (and this is not all that uncommon), you will need to run and examine the individual steps to carry out useful debugging.

30.1. Introduction

This section describes the use of the python script MMPBSA.py [589] to perform Molecular Mechanics / Poisson Boltzmann (or Generalized Born) Surface Area (MM/PB(GB)SA) calculations. This is a post-processing method in which representative snapshots from an ensemble of conformations are used to calculate the free energy change between two states (typically a bound and free state of a receptor and ligand). Free energy differences are calculated by combining the so-called gas phase energy contributions that are independent of the chosen solvent model as well as solvation free energy components (both polar and non-polar) calculated from an implicit solvent model for each species. Entropy contributions to the total free energy may be added as a further refinement. The entropy calculations can be done in either a HCT Generalized Born solvation model [141, 152] or in the gas phase using a *mmpbsa_py_nabnmode* program written in the *nab* programming language, or via the quasi-harmonic approximation in *ptraj*.

The gas phase free energy contributions are calculated by *sander* within the Amber program suite or *mmpbsa_py_energy* within the AmberTools package according to the force field with which the topology files were created. The solvation free energy contributions may be further decomposed into an electrostatic and hydrophobic contribution. The electrostatic portion is calculated using the Poisson Boltzmann (PB) equation, the Generalized Born method, or the Reference Interaction Site Model (RISM). The PB equation is solved numerically by either the *pbsa* program included with AmberTools or by the Adaptive Poisson Boltzmann Solver (APBS) program through the iAPBS interface[377] with Amber (for more information, see http://www.poissonboltzmann.org/apbs). The hydrophobic contribution is approximated by the LCPO method [126] implemented within *sander* or the *molsurf* method as implemented in *cpptraj*.

MM/PB(GB)SA typically employs the approximation that the configurational space explored by the systems are very similar between the bound and unbound states, so every snapshot for each species is extracted from the same trajectory file, although MMPBSA.py will accept separate trajectory files for each species. Furthermore, explicit solvent and ions are stripped from the trajectory file(s) to hasten convergence by preventing solvent-solvent interactions from dominating the energy terms. A more detailed explanation of the theory can be found in Srinivasan, et. al.[590] You may also wish to refer to reviews summarizing many of the applications of this model,[591, 592] as well as to papers describing some of its applications.[593–597]

Many popular types of MM/PBSA calculations can be performed using just AmberTools, while some of the more advanced functionality requires the *sander* program from Amber.

30.2. Preparing for an MM/PB(GB)SA calculation

MM/PB(GB)SA is often a very useful tool for obtaining relative free energies of binding when comparing ligands. Perhaps its biggest advantage is that it is very computationally inexpensive compared to other free energy calculations, such as TI or FEP. Following the advice given below before any MD simulations are run will make running MMPBSA.py successfully much easier.

30.2.1. Building Topology Files

MMPBSA.py requires at least three, usually four, compatible topology files. If you plan on running MD in explicit water, you will need a solvated topology file of the entire complex, and you will always need a topology for the entire complex, one for just the receptor, and a final one for just the ligand. Moreover, they must be compatible with one another (i.e., each must have the same charges for the same atoms, the same force field must be used for all three of the required prmtops, and they must have the same PBRadii set, see LEaP for description of pbradii). Thus, it is strongly advised that all prmtop files are created with the same script. We run through a typical example here, though leave some of the details to other sections and other tutorials. We will start with a system that is a large protein binding a small, one-residue ligand. We will assume that a docked structure has already been obtained as a PDB and that two separate PDBs have been constructed, receptor.pdb and LIG.pdb. We will also assume that a MOL2 file was created from LIG.pdb, residue name 'LIG', was built with charges already derived (either through antechamber or some other method), and an frcmod file for 'LIG' that contains all missing parameters have already been created. Furthermore, we will use the FF14SB force field for this example. A sample script file called, for instance, mmpbsa_leap.in, is shown below

```
source leaprc.protein.ff14SB
source leaprc.water.tip3p
loadAmberParams LIG.frcmod
LIG = loadMol2 LIG.mol2
receptor = loadPDB receptor.pdb
complex = combine {receptor LIG}
set default PBRadii mbondi2

saveAmberParm LIG lig.top lig.crd
saveAmberParm receptor rec.top rec.crd
saveAmberParm complex com.top com.crd

solvateOct complex TIP3PBOX 15.0
saveAmberParm complex com_solvated.top com_solvated.crd
quit
```

The above script, when executed using the command

```
tleap -f mmpbsa_leap.in
```

should produce four prmtop files, lig.top, rec.top, com.top, and com_solvated.top. Topology files created in this manner will make running MMPBSA.py far easier. This is, of course, the simplest case, but we briefly describe some other examples. MMPBSA.py will guess the mask for both the receptor and ligand inside the complex topology file as long as the ligand residues appear continuously in the complex topology file. Therefore, if you're adding two ligands, combine them consecutively in the complex (rather than one residue at the beginning and one at the end, for instance). If you have done this, you should allow MMPBSA.py to guess the masks since it provides a good error check.

30.2.2. Using ante-MMPBSA.py

ante-MMPBSA.py is a python utility that allows you to create compatible complex, receptor, and ligand topology files from a solvated topology file, or compatible receptor and ligand topology files from a complex topology file. The usage statement for ante-MMPBSA.py is

```
Usage: ante-MMPBSA.py [options]
Options:
  -h, --help            show this help message and exit
  -p PRMTOP, --prmtop=PRMTOP
                        Input "dry" complex topology or solvated complex
                        topology
  -c COMPLEX, --complex-prmtop=COMPLEX
                        Complex topology file created by stripping PRMTOP of
                        solvent
  -r RECEPTOR, --receptor-prmtop=RECEPTOR
                        Receptor topology file created by stripping COMPLEX of
                        ligand
  -l LIGAND, --ligand-prmtop=LIGAND
                        Ligand topology file created by stripping COMPLEX of
                        receptor
  -s STRIP_MASK, --strip-mask=STRIP_MASK
                        Amber mask of atoms needed to be stripped from PRMTOP
                        to make the COMPLEX topology file
  -m RECEPTOR_MASK, --receptor-mask=RECEPTOR_MASK
                        Amber mask of atoms needed to be stripped from COMPLEX
                        to create RECEPTOR. Cannot specify with -n/--ligand-
                        mask
  -n LIGAND_MASK, --ligand-mask=LIGAND_MASK
                        Amber mask of atoms needed to be stripped from COMPLEX
                        to create LIGAND. Cannot specify with -m/--receptor-
                        mask
  --radii=RADIUS_SET    PB/GB Radius set to set in the generated topology
                        files. This is equivalent to "set PBRadii <radius>" in
                        LEaP. Options are bondi, mbondi2, mbondi3, amber6, and
                        mbondi and the default is to use the existing radii.
```

The input prmtop is required. It can either be a solvated, complex topology file or a complex topology file with no solvent present. If a strip_mask is given, you must also provide a complex topology file, and that complex topology file will be created by stripping strip_mask from the input prmtop. If you wish to create receptor and ligand topology files (you must create both or neither), provide BOTH a –receptor-prmtop and a –ligand-prmtop file name, as well as only ONE of either –receptor-mask or –ligand-mask. Whichever mask you do NOT define will be defined as the negated mask that you DID provide.

In short, you can use ante-MMPBSA.py to strip solvent from your prmtop for 3 applications.

1. Strip solvent from a solvated topology file and write out a non-solvated topology file.
2. Create ligand and receptor topologies from a complex topology by removing a given ligand or receptor mask.
3. A combination of 1 and 2 in the same command.

30.2.3. Running Molecular Dynamics

Not many details will be given here because MM/PB(GB)SA is a post-processing trajectory analysis technique. Molecular dynamics are run to generate an ensemble of snapshots upon which to calculate the binding energy. This technique is most effective when the structures are not correlated, which means that the simulated time between extracted snapshots should be sufficiently large to avoid such correlation.

There are two techniques that can be employed when running these simulations with respect to MMPBSA.py. The first is what's called the "single trajectory protocol" and the second of which is called the "multiple trajectory protocol". The first method will extract the snapshots for the complex, receptor, and ligand from the same trajectory. This is a faster method because it requires the simulation of only a single system, but makes the assumption that the configurational space explored by the receptor and ligand is unchanged between the bound and unbound states. The latter method eliminates this assumption at the cost of more simulations. MMPBSA.py requires a complex trajectory, but will accept a receptor and/or ligand trajectory as well. Any trajectory not given to the script will be extracted from the complex trajectory.

30.3. Running MMPBSA.py

30.3.1. The input file

The input file was designed to be as syntactically similar to other programs in Amber as possible. The input file has the same namelist structure as both *sander* and *pmemd*. The allowed namelists are &general, &gb, &pb, &rism, &alanine_scanning, &nmode, and &decomp. The input variables recognized in each namelist are described below, but those in &general are typically variables that apply to all aspects of the calculation. The &gb namelist is unique to Generalized Born calculations, &pb is unique to Poisson Boltzmann calculations, &rism is unique to 3D-RISM calculations, &alanine_scanning is unique to alanine scanning calculations, &nmode is unique to the normal mode calculations used to approximate vibrational entropies, and &decomp is unique to the decomposition scheme. All of the input variables are described below according to their respective namelists. Integers and floating point variables should be typed as-is while strings should be put in either single- or double-quotes. All variables should be set with “variable = value” and separated by commas. See the examples below. Variables will usually be matched to the minimum number of characters required to uniquely identify that variable within that namelist. Variables require at least 4 characters to be matched unless that variable name has fewer than 4 characters (in which case the whole variable name is required). For example, “star” in &general will match “startframe”. However, “stare” and “sta” will match nothing.

&general namelist variables

debug_printlevel MMPBSA.py prints errors by raising exceptions, and not catching fatal errors. If debug_printlevel is set to 0, then detailed tracebacks (effectively the call stack showing exactly where in the program the error occurred) is suppressed, so only the error message is printed. If debug_printlevel is set to 1 or higher, all tracebacks are printed, which aids in debugging of issues. Default: 0. (Advanced Option)

endframe The frame from which to stop extracting snapshots from the full, concatenated trajectory comprised of every trajectory file supplied on the command-line. (Default = 9999999)

entropy Specifies whether or not a quasi-harmonic entropy approximation is made with ptraj. Allowed values are 0: Don’t. 1: Do (Default = 0)

interval The offset from which to choose frames from each trajectory file. For example, an interval of 2 will pull every 2nd frame beginning at startframe and ending less than or equal to endframe. (Default = 1)

keep_files The variable that specifies which temporary files are kept. All temporary files have the prefix “_MMPBSA_” prepended to them (unless you change the prefix on the command-line—see subsection Subsection 30.3.2 for details). Allowed values are 0, 1, and 2.
0: Keep no temporary files
1: Keep all generated trajectory files and mdout files created by sander simulations
2: Keep all temporary files. Temporary files are only deleted if MMPBSA.py completes successfully
(Default = 1) A verbose level of 1 is sufficient to use -rewrite-output and recreate the output file without rerunning any simulations.

ligand_mask The mask that specifies the ligand residues within the complex prmtop (NOT the solvated prmtop if there is one). The default guess is generally sufficient and will only fail as stated above. You should use the default mask assignment if possible because it provides a good error catch. This follows the same description as the receptor_mask above.

netcdf Specifies whether or not to use NetCDF trajectories internally rather than writing temporary ASCII trajectory files. NOTE: NetCDF trajectories can be used as input for MMPBSA.py regardless of what this variable is set to, but NetCDF trajectories are faster to write and read. For very large trajectories, this could offer significant speedups, and requires less temporary space. However, this option is incompatible with alanine scanning. Default value is 0.
0: Do NOT use temporary NetCDF trajectories
1: Use temporary NetCDF trajectories

receptor_mask The mask that specifies the receptor residues within the complex prmtop (NOT the solvated prmtop if there is one). The default guess is generally sufficient and will only fail if the ligand residues are not found in succession within the complex prmtop. You should use the default mask assignment if possible because it provides a good error catch. It uses the "Amber mask" syntax described elsewhere in this manual. This will be replaced with the default receptor_mask if ligand_mask (below) is not also set.

search_path Advanced option. By default, MMPBSA.py will only search for executables in `$AMBERHOME/bin`. To enable it to search for binaries in your full `PATH` if they can't be found in `$AMBERHOME/bin`, set search_path to 1. Default 0 (do not search through the `PATH`). This is particularly useful if you are using an older version of *sander* that is not in `AMBERHOME`.

startframe The frame from which to begin extracting snapshots from the full, concatenated trajectory comprised of every trajectory file placed on the command-line. This is always the first frame read. (Default = 1)

strip_mask The variable that specifies which atoms are stripped from the trajectory file if a *solvated_prmtop* is provided on the command-line. See 30.3.2. (Default = ":WAT:CL:CIO:CS:IB:K:LI:MG:NA:RB")

use_sander Forces MMPBSA.py to use *sander* for energy calculations, even when *mmpbsa_py_energy* will suffice (Default 0)
0 - Use *mmpbsa_py_energy* when possible
1 - Always use *sander*

full_traj This variable is for calculations performed in parallel to control whether complete trajectories are made of the complex, receptor, and ligand. In parallel calculations, a different trajectory is made for each processor to analyze only the selected frames for that processor. A value of 0 will only create the intermediate trajectories analyzed by each processor, while a value of 1 will additionally combine those trajectories to make a single trajectory of all frames analyzed across all processors for the complex, receptor, and ligand. (Default = 0)

verbose The variable that specifies how much output is printed in the output file. There are three allowed values: 0, 1, and 2. A value of 0 will simply print difference terms, 1 will print all complex, receptor, and ligand terms, and 2 will also print bonded terms if one trajectory is used. (Default = 1)

&gb namelist variables

ifqnt Specifies whether a part of the system is treated with quantum mechanics. 1: Use QM/MM, 0: Potential function is strictly classical (Default = 0). This functionality requires *sander*

igb Generalized Born method to use (seeSection 4). Allowed values are 1, 2, 5, 7 and 8. (Default = 5) All models are now available with both *mmpbsa_py_energy* and *sander*

qm_residues Comma- or semicolon-delimited list of complex residues to treat with quantum mechanics. All whitespace is ignored. All residues treated with quantum mechanics in the complex must be treated with quantum mechanics in the receptor or ligand to obtain meaningful results. If the default masks are used, then MMPBSA.py will figure out which residues should be treated with QM in the receptor and ligand. Otherwise, skeleton mdin files will be created and you will have to manually enter qmmask in the ligand and receptor topology files. There is no default, this must be specified.

qm_theory Which semi-empirical Hamiltonian should be used for the quantum calculation. No default, this must be specified. See its description in the QM/MM section of the manual for options.

qmcharge_com The charge of the quantum section for the complex. (Default = 0)

qmcharge_lig The charge of the quantum section of the ligand. (Default = 0)

qmcharge_rec The charge of the quantum section for the receptor. (Default = 0)

qmcut The cutoff for the qm/mm charge interactions. (Default = 9999.0)

saltcon Salt concentration in Molarity. (Default = 0.0)

surfoff Offset to correct (by addition) the value of the non-polar contribution to the solvation free energy term (Default 0.0)

surften Surface tension value (Default = 0.0072). Units in $kcal/mol/\AA^2$

molsurf When set to 1, use the molsurf algorithm to calculate the surface area for the nonpolar solvation term. When set to 0, use LCPO (Linear Combination of Pairwise Overlaps). (Default 0)

probe Radius of the probe molecule (supposed to be the size of a solvent molecule), in Angstroms, to use when determining the molecular surface (only applicable when molsurf is set to 1). Default is 1.4.

msoffset Offset to apply to the individual atomic radii in the system when calculating the molsurf surface. See the description of the molsurf action command in *cpptraj*. Default is 0.

&pb namelist variables

inp Nonpolar optimization method (Default = 2).

cavity_offset Offset value used to correct non-polar free energy contribution (Default = -0.5692) This is not used for APBS.

cavity_surften Surface tension. (Default = 0.0378 kcal/mol Angstrom^2). Unit conversion to *kJ* done automatically for APBS.

exdi External dielectric constant (Default = 80.0).

indi Internal dielectric constant (Default = 1.0).

fillratio The ratio between the longest dimension of the rectangular finite-difference grid and that of the solute (Default = 4.0).

scale Resolution of the Poisson Boltzmann grid. It is equal to the reciprocal of the grid spacing. (Default = 2.0)

istrng Ionic strength in Molarity. It is converted to mM for PBSA and kept as M for APBS. (Default = 0.0)

linit Maximum number of iterations of the linear Poisson Boltzmann equation to try (Default = 1000)

prbrad Solvent probe radius in Angstroms. Allowed values are 1.4 and 1.6 (Default = 1.4)

radiopt The option to set up atomic radii according to 0: the prmtop, or 1: pre-computed values (see Amber manual for more complete description). (Default = 1)

sander_apbs Option to use APBS for PB calculation instead of the built-in PBSA solver. This will work only through the iAPBS interface[377] built into sander.APBS. Instructions for this can be found online at the iAPBS/APBS websites. Allowed values are 0: Don't use APBS, or 1: Use sander.APBS. (Default = 0)

memopt Turn on membrane protein support (Default = 0).

emem Membrane dielectric constant (Default = 1.0).

mthick Membrane thickness (Default = 40.0).

mctrdz Absolute membrane center in the z-direction (Default=0.0, use protein center as the membrane center).

poretype Turn on the automatic membrane channel/pore finding method (Default=1).

A more thorough description of these and other options can be found in Chapter 6.

&alanine_scanning namelist variables

mutant_only Option to perform specified calculations only for the mutants. Allowed values are 0: Do mutant and original or 1: Do mutant only (Default = 0)

Note that all calculation details are controlled in the other namelists, though for alanine scanning to be performed, the namelist must be included (blank if desired)

&nmode namelist variables

dielc Distance-dependent dielectric constant (Default = 1.0)

drms Convergence criteria for minimized energy gradient. (Default = 0.001)

maxcyc Maximum number of minimization cycles to use per snapshot in sander. (Default = 10000)

nminterval* Offset from which to choose frames to perform nmode calculations on (Default = 1)

nmendframe* Frame number to stop performing nmode calculations on (Default = 1000000)

nmode_igb Value for Generalized Born model to be used in calculations. Options are 0: Vacuum, 1: HCT GB model [141, 152] (Default 1)

nmode_istrng Ionic strength to use in nmode calculations. Units are Molarity. Non-zero values are ignored if *nmode_igb* is 0 above. (Default = 0.0)

nmstartframe* Frame number to begin performing nmode calculations on (Default = 1)

* These variables will choose a subset of the frames chosen from the variables in the &general namelist. Thus, the "trajectory" from which snapshots will be chosen for nmode calculations will be the collection of snapshots upon which the other calculations were performed.

&decomp namelist variables

csv_format Print the decomposition output in a Comma-Separated-Variable (CSV) file. CSV files open natively in most spreadsheets. If set to 1, this variable will cause the data to be written out in a CSV file, and standard error of the mean will be calculated and included for all data. If set to 0, the standard, ASCII format will be used for the output file. Default is 1 (CSV-formatted output file)

dec_verbose Set the level of output to print in the decmop_output file.
0 - DELTA energy, total contribution only
1 - DELTA energy, total, sidechain, and backbone contributions
2 - Complex, Receptor, Ligand, and DELTA energies, total contribution only
3 - Complex, Receptor, Ligand, and DELTA energies, total, sidechain, and backbone contributions
Note: If the values 0 or 2 are chosen, only the Total contributions are required, so only those will be printed to the mdout files to cut down on the size of the mdout files and the time required to parse them. However, this means that -rewrite-output cannot be used to change the default verbosity to print out sidechain and/or backbone energies, but it can be used to reduce the amount of information printed to the final output. The parser will extract as much information from the mdout files as it can, but will complain and quit if it cannot find everything it's being asked for.
Default = 0

idecomp Energy decomposition scheme to use:
1 - Per-residue decomp with 1-4 terms added to internal potential terms
2 - Per-residue decomp with 1-4 EEL added to EEL and 1-4 VDW added to VDW potential terms.
3 - Pairwise decomp with 1-4 terms added to internal potential terms
4 - Pairwise decomp with 1-4 EEL added to EEL and 1-4 VDW added to VDW potential terms
(No default. This must be specified!) This functionality requires *sander*.

print_res Select residues from the complex prmtop to print. The receptor/ligand residues will be automatically figured out if the default mask assignments are used. If you specify your own masks, you will need to modify the mdin files created by MMPBSA.py and rerun MMPBSA.py with the -use-mdins flag. Note that the DELTAs will not be computed in this case. This variable accepts a sequence of individual residues and/or ranges. The different fields must be either comma- or semicolon-delimited. For example: print_res = "1, 3-10, 15, 100", or print_res = "1; 3-10; 15; 100". Both of these will print residues 1, 3 through 10, 15, and 100 from the complex prmtop and the corresponding residues in either the ligand and/or receptor prmtops. (Default: print all residues)*

* Please note: Using idecomp=3 or 4 (pairwise) with a very large number of printed residues and a large number of frames can quickly create very, very large temporary mdout files. Large print selections also demand a large amount of memory to parse the mdout files and write decomposition output file (~500 MB for just 250 residues, since that's 62500 pairs!) It is not unusual for the output file to take a significant amount of time to print if you have a lot of data. This is most applicable to pairwise decomp, since the amount of data scales as $O(N^2)$.

&rism namelist variables*

buffer Minimum distance between solute and edge of solvation box. Specify this with grdspc below. Mutually exclusive with ng and solvbox. Set buffer < 0 if you wish to use ng and solvbox. (Default = 14 Å)

closure The approximation to the closure relation. Allowed choices are *kh* (Kovalenko-Hirata), *hnc* (Hypernetted-chain), or *psen* (Partial Series Expansion of order-*n*) where "*n*" is a positive integer (e.g., "pse3"). (Default = 'kh')

closureorder (Deprecated) The order at which the PSE-*n* closure is truncated if closure is specified as "pse" or "psen" (no integers). (Default = 1)

grdspc Grid spacing of the solvation box. Specify this with buffer above. Mutually exclusive with ng and solvbox. (Default = 0.5 Å)

ng Number of grid points to use in the x, y, and z directions. Used only if buffer < 0. Mutually exclusive with buffer and grdspc above, and paired with solvbox below. No default, this must be set if buffer < 0. Define like "ng=1000,1000,1000"

polardecomp Decompose the solvation free energy into polar and non-polar contributions. Note that this will increase computation time by roughly 80%. 0: Don't decompose solvation free energy. 1: Decompose solvation free energy. (Default = 0)

rism_verbose Level of output in temporary RISM output files. May be helpful for debugging or following convergence. Allowed values are 0 (just print the final result), 1 (additionally prints the total number of iterations for each solution), and 2 (additionally prints the residual for each iteration and details of the MDIIS solver). (Default = 0)

solvbox Length of the solvation box in the x, y, and z dimensions. Used only if buffer < 0. Mutually exclusive with buffer and grdspc above, and paired with ng above. No default, this must be set if buffer < 0. Define like "solvbox=20,20,20"

solvcut Cutoff used for solute-solvent interactions. The default is the value of buffer. Therefore, if you set buffer < 0 and specify ng and solvbox instead, you must set solvcut to a nonzero value or the program will quit in error. (Default = buffer)

thermo Which thermodynamic equation you want to use to calculate solvation properties. Options are “std”, “gf”, or “both” (case-INsensitive). “std” uses the standard closure relation, “gf” uses the Gaussian Fluctuation approximation, and “both” will print out separate sections for both. (Default = “std”). Note that all data are printed out for each RISM simulation, so no choice is any more computationally demanding than another. Also, you can change this option and use the -rewrite-output flag to obtain a different printout after-the-fact.

tolerance Upper bound of the precision requirement used to determine convergence of the self-consistent solution. This has a strong effect on the cost of 3D-RISM calculations. (Default = 1e-5).

* 3D-RISM calculations are performed with the rism3d.snglpnt program built with AmberTools, written by Tyler Luchko. It is the most expensive, yet most statistical mechanically rigorous solvation model available in MMPBSA.py. See Chapter 7 for a more thorough description of options and theory. A list of references can be found there, too. One advantage of 3D-RISM is that an arbitrary solvent can be chosen; you just need to change the xvvfile specified on the command line (see 30.3.2).

Sample input files

```
Sample input file for GB and PB calculation
&general
   startframe=5, endframe=100, interval=5,
   verbose=2, keep_files=0,
/
&gb
   igb=5, saltcon=0.150,
/
&pb
   istrng=0.15, fillratio=4.0
/
--------------------------------------------------------
Sample input file for Alanine scanning
&general
   verbose=2,
/
&gb
   igb=2, saltcon=0.10
/
&alanine_scanning
/
--------------------------------------------------------
Sample input file with nmode analysis
&general
   startframe=5, endframe=100, interval=5,
   verbose=2, keep_files=2,
/
&gb
   igb=5, saltcon=0.150,
/
&nmode
   nmstartframe=2, nmendframe=20, nminterval=2,
   maxcyc=50000, drms=0.0001,
/
--------------------------------------------------------
Sample input file with decomposition analysis
&general
```

```
   startframe=5, endframe=100, interval=5,
/
&gb
   igb=5, saltcon=0.150,
/
&decomp
   idecomp=2, dec_verbose=3,
   print_res="20, 40-80, 200"
/
--------------------------------------------------------
Sample input file for QM/MMGBSA
&general
   startframe=5, endframe=100, interval=5,
/
&gb
   igb=5, saltcon=0.100, ifqnt=1, qmcharge=0,
   qm_residues="100-105, 200", qm_theory="PM3"
/
--------------------------------------------------------
Sample input file for MM/3D-RISM
&general
   startframe=5, endframe=100, interval=5,
/
&rism
   polardecomp=1, thermo="gf"
/
--------------------------------------------------------
Sample input file for MMPBSA with membrane proteins
&general
   use_sander=1,
   startframe=1, endframe=100, interval=1,
   keep_files=0, debug_printlevel=2
/
&pb
   radiopt=0, indi=20.0, istrng=0.150,
   fillratio=1.25, ipb=1, nfocus=1,
   bcopt=10, eneopt=1, cutfd=7.0, cutnb=99.0,
   npbverb=1, solvopt=2, inp=1,
   memopt=1, emem=7.0, mctrdz=-10.383, mthick=36.086, poretype=1,
   maxarcdot=15000
/
```

A few important notes about input files. Comments are allowed by placing a # at the beginning of the line (whitespace is ignored). Variable initialization may span multiple lines. In-line comments (i.e., putting a # for a comment after a variable is initialized in the same line) is not allowed and will result in an input error. Variable declarations must be comma-delimited, though all whitespace is ignored. Finally, all lines between namelists are ignored, so comments may be put before each namelist without using #.

30.3.2. Calling MMPBSA.py from the command-line

MMPBSA.py is invoked through the command line as follows:

```
Usage: MMPBSA.py  [Options]
Options:
```

```
--help, -h, --h, -H
      show this help message and exit
-O
      Overwrite existing output files
-i        input_file
      MM/PBSA input file
-o         output_file
      Final MM/PBSA statistics file. Default
      FINAL_RESULTS_MMPBSA.dat
-sp        solvated_prmtop
      Solvated complex topology file
-cp        complex_prmtop
      Complex topology file. Default "complex_prmtop"
-rp        receptor_prmtop
      Receptor topology file
-lp        ligand_prmtop
      Ligand topology file
-y         mdcrd1,mdcrd2,...,mdcrdN
      Input trajectories to analyze. Default mdcrd
-do        decompout
      Decomposition statistics summary file. Default
      FINAL_DECOMP_MMPBSA.dat
-eo        energyout
      CSV-format output of all energy terms for every frame in
      every calculation. File name forced to end in .csv
-deo       dec_energies
      CSV-format output of all decomposition energy terms for
      every frame. File name forced to end in .csv
-yr        receptor_mdcrd1,receptor_mdcrd2,...,receptor_mdcrdN
      Receptor trajectory file for multiple trajectory approach
-yl        ligand_mdcrd1,ligand_mdcrd2,...,ligand_mdcrdN
      Ligand trajectory file for multiple trajectory approach
-mc        mutant_complex_prmtop
      Alanine scanning mutant complex topology file
-ml        mutant_ligand_prmtop
      Alanine scanning mutant ligand topology file
-mr        mutant_receptor_prmtop
      Alanine scanning mutant receptor topology file
-slp       solvated_ligand_prmtop
      Solvated ligand topology file
-srp       solvated_receptor_prmtop
      Solvated receptor topology file
-xvvfile   xvvfile
      XVV file for 3D-RISM. Default
      $AMBERHOME/dat/mmpbsa/spc.xvv
-prefix    prefix
      Beginning of every intermediate file name generated
-make-mdins
      Create the Input files for each calculation and quit
-use-mdins
      Use existing input files for each calculation
-rewrite-output
      Don't rerun any calculations, just parse existing output
```

```
                    files
            --clean
                    Clean temporary files from previous run
```

`-make-mdins` and `-use-mdins` are intended to give added flexibility to user input. If the MM/PBSA input file does not expose a variable you require, you may use the `-make-mdins` flag to generate the MDIN files and then quit. Then, edit those MDIN files, changing the variables you need to, then running MMPBSA.py with `-use-mdins` to use those modified files.

`--clean` will remove all temporary files created by MMPBSA.py in a previous calculation.

`--version` will display the program version and exit.

30.3.3. Running MMPBSA.py

30.3.3.1. Serial version

This version is installed with Amber during the serial install of AmberTools. `AMBERHOME` must be set, or it will quit on error. If any changes are made to the modules, MMPBSA.py must be remade so the updated modules are found by MMPBSA.py. An example command-line call is shown below:

```
MMPBSA.py -O -i mmpbsa.in -cp com.top -rp rec.top -lp lig.top -y traj.crd
```

The tests, found in `${AMBERHOME}/test/mmpbsa_py` provide good examples for running MMPBSA.py calculations.

30.3.3.2. Parallel (MPI) version

This version is installed with Amber during the parallel install. The python package mpi4py is included with the MMPBSA.py source code and must be successfully installed in order to run the MPI version of MMPBSA.py. It is run in the same way that the serial version is above, except MPI directions must be given on the command line as well. Note, if `mpi4py` does not install correctly, you must install it yourself in order to use MMPBSA.py.MPI. One note: at a certain level, running RISM in parallel may actually hurt performance, since previous solutions are used as an initial guess for the next frame, hastening convergence. Running in parallel loses this advantage. Also, due to the overhead involved in which each thread is required to load every topology file when calculating energies, parallel scaling will begin to fall off as the number of threads reaches the number of frames. A usage example is shown below:

```
mpirun -np 2 MMPBSA.py.MPI -O -i mmpbsa.in -cp com.top -rp rec.top \
                          -lp lig.top -y traj.crd
```

30.3.4. Types of calculations you can do

There are many different options for running MMPBSA.py. Among the types of calculations you can do are:

1. Normal binding free energies, with either PB or GB implicit solvent models. Each can be done with either 1, 2, or 3 different trajectories, but the complex, receptor, and ligand topology files must all be defined. The complex mdcrd must always be provided. Whichever trajectories of the receptor and/or ligand that are NOT specified will be extracted from the complex trajectory. This allows a 1-, 2-, or 3-trajectory analysis. All PB calculations and GB models can be performed with just AmberTools via the *mmpbsa_py_energy* program installed with MMPBSA.py.

2. Stability calculations with any calculation type. If you only specify the complex prmtop (and leave receptor and ligand prmtop options blank), then a “stability” calculation will be performed, and you will get statistics based on only a single system. Any additional receptor or ligand information given will be ignored, but note that if receptor and/or ligand topologies are given, it will no longer be considered a stability calculation. The previous statement refers principally to mutated receptor/ligand files or extra ligand/receptor trajectory files.

3. Alanine scanning with either PB or GB implicit solvent models. All trajectories will be mutated to match the mutated topology files, and whichever calculations that would be carried out for the normal systems are also carried out for the mutated systems. Note that only 1 mutation is allowed per simulation, and it must be to an alanine. If mutant_only is not set to 1, differences resulting from the mutations are calculated. This option is incompatible with intermediate NetCDF trajectories (see the netcdf = 1 option above). This has the same program requirements as option 1 above.
4. Entropy corrections. An entropy term can be added to the free energies calculated above using either the quasi-harmonic approximation or the normal mode approximation. Calculations will be done for the normal and mutated systems (alanine scanning) as requested. Normal mode calculations are done with the *mmpbsa_py_nabnmode* program included with AmberTools.
5. Decomposition schemes. The energy terms will be decomposed according to the decomposition scheme outlined in the idecomp variable description. This should work with all of the above, though entropy terms cannot be decomposed. APBS energies cannot be decomposed, either. Neither can PBSA surface area terms. This functionality requires *sander* from the Amber 11 (or later) package.
6. QM/MMGBSA. This is a binding free energy (or stability calculation) using the Generalized Born solvent model allowing you to treat part of your system with a quantum mechanical Hamiltonian. See “Advanced Options” for tips about optimizing this option. This functionality requires *sander* from the Amber package.
7. MM/3D-RISM. This is a binding free energy (or stability calculation) using the 3D-RISM solvation model. This functionality is performed with *rism3d.snglpnt* built with AmberTools.
8. Membrane Protein MMPBSA. Calculate the MMPBSA binding free energy for a ligand bound to a protein that is embedded into a membrane. Only use_sander=1 is supported.

30.3.5. The Output File

The header of the output file will contain information about the calculation. It will show a copy of the input file as well as the names of all files that were used in the calculation (topology files and coordinate file(s)). If the masks were not specified, it prints its best guess so that you can verify its accuracy, along with the residue name of the ligand (if it is only a single residue).

The energy and entropy contributions are broken up into their components as they are in *sander* and *nmode* or *ptraj*. The contributions are further broken into G_{gas} and G_{solv}. The polar and non-polar contributions are EGB (or EPB) and ESURF (or ECAVITY / ENPOLAR), respectively for GB (or PB) calculations.

By default, bonded terms are not printed for any one-trajectory simulation. They are computed and their differences calculated, however. They are not shown (nor included in the total) unless specifically asked for because they should cancel completely. A single trajectory does not produce any differences between bond lengths, angles, or dihedrals between the complex and receptor/ligand structures. Thus, when subtracted they cancel completely. This includes the BOND, ANGLE, DIHED, and 1-4 interactions. If inconsistencies are found, these values are displayed and inconsistency warnings are printed. When this occurs the results are generally useless. Of course this does not hold for the multiple trajectory protocol, and so all energy components are printed in this case.

Finally, all warnings generated during the calculation that do not result in fatal errors are printed after calculation details but before any results.

30.3.6. Temporary Files

MMPBSA.py creates working files during the execution of the script beginning with the prefix _MMPBSA_. The variable “keep_files” controls how many of these files are kept after the script finishes successfully. If the script quits in error, all files will be kept. You can clean all temporary files from a directory by running MMPBSA –clean described above.

If MMPBSA.py does not finish successfully, several of these files may be helpful in diagnosing the problem. For that reason, every temporary file is described below. Note that not every temporary file is generated in every simulation. At the end of each description, the lowest value of “keep_files” that will retain this file will be shown in parentheses.

_MMPBSA_gb.mdin Input file that controls the GB calculation done in *sander*. (2)

_MMPBSA_pb.mdin Input file that controls the PB calculation done in *sander*. (2)

_MMPBSA_gb_decomp_com.mdin Input file that controls the GB decomp calculation for the complex done in *sander.* (2)

_MMPBSA_gb_decomp_rec.mdin Input file that controls the GB decomp calculation for the receptor done in *sander*. (2)

_MMPBSA_gb_decomp_lig.mdin Input file that controls the GB decomp calculation for the ligand done in *sander*. (2)

_MMPBSA_pb_decomp_com.mdin Input file that controls the PB decomp calculation for the complex done in *sander*. (2)

_MMPBSA_pb_decomp_rec.mdin Input file that controls the PB decomp calculation for the receptor done in *sander*. (2)

_MMPBSA_pb_decomp_lig.mdin Input file that controls the PB decomp calculation for the ligand done in *sander*. (2)

_MMPBSA_gb_qmmm_com.mdin Input file that controls the GB QM/MM calculation for the complex done in *sander*. (2)

_MMPBSA_gb_qmmm_rec.mdin Input file that controls the GB QM/MM calculation for the receptor done in *sander*. (2)

_MMPBSA_gb_qmmm_lig.mdin Input file that controls the GB QM/MM calculation for the ligand done in *sander.* (2)

_MMPBSA_complex.mdcrd.# Trajectory file(s) that contains only those complex snapshots that will be processed by MMPBSA.py. (1)

_MMPBSA_ligand.mdcrd.# Trajectory file(s) that contains only those ligand snapshots that will be processed by MMPBSA.py. (1)

_MMPBSA_receptor.mdcrd.# Trajectory file(s) that contains only those receptor snapshots that will be processed by MMPBSA.py. (1)

_MMPBSA_complex_nc.# Same as _MMPBSA_complex.mdcrd.#, except in the NetCDF format. (1)

_MMPBSA_receptor_nc.# Same as _MMPBSA_receptor.mdcrd.#, except in the NetCDF format. (1)

_MMPBSA_ligand_nc.# Same as _MMPBSA_ligand.mdcrd.#, except in the NetCDF format. (1)

_MMPBSA_dummycomplex.inpcrd Dummy inpcrd file generated by _MMPBSA_complexinpcrd.in for use with imin=5 functionality in *sander*. (1)

_MMPBSA_dummyreceptor.inpcrd Same as above, but for the receptor. (1)

_MMPBSA_dummyligand.inpcrd Same as above, but for the ligand. (1)

_MMPBSA_complex.pdb Dummy PDB file of the complex required to set molecule up in nab programs

_MMPBSA_receptor.pdb Dummy PDB file of the receptor required to set molecule up in nab programs

_MMPBSA_ligand.pdb Dummy PDB file of the ligand required to set molecule up in nab programs

_MMPBSA_complex_nm.mdcrd.# Trajectory file(s) for each thread with snapshots used for normal mode calculations on the complex. (1)

_MMPBSA_receptor_nm.mdcrd.# Trajectory file for each thread with snapshots used for normal mode calculations on the receptor. (1)

_MMPBSA_ligand_nm.mdcrd.# Trajectory file for each thread with snapshots used for normal mode calculations on the ligand. (1)

_MMPBSA_ptrajentropy.in Input file that calculates the entropy via the quasi-harmonic approximation. This file is processed by *ptraj*. (2)

_MMPBSA_avgcomplex.pdb PDB file containing the average positions of all complex conformations processed by _MMPBSA_cenptraj.in. It is used as the reference for the _MMPBSA_ptrajentropy.in file above. (1)

_MMPBSA_complex_entropy.out File into which the entropy results from _MMPBSA_ptrajentropy.in analysis on the complex are dumped. (1)

_MMPBSA_receptor_entropy.out Same as above, but for the receptor. (1)

_MMPBSA_ligand_entropy.out Same as above, but for the ligand. (1)

_MMPBSA_ptraj_entropy.out Output from running *ptraj* using _MMPBSA_ptrajentropy.in. (1)

_MMPBSA_complex_gb.mdout.# *sander* output file containing energy components of all complex snapshots done in GB. (1)

_MMPBSA_receptor_gb.mdout.# *sander* output file containing energy components of all receptor snapshots done in GB. (1)

_MMPBSA_ligand_gb.mdout.# *sander* output file containing energy components of all ligand snapshots done in GB. (1)

_MMPBSA_complex_pb.mdout.# *sander* output file containing energy components of all complex snapshots done in PB. (1)

_MMPBSA_receptor_pb.mdout.# *sander* output file containing energy components of all receptor snapshots done in PB. (1)

_MMPBSA_ligand_pb.mdout.# *sander* output file containing energy components of all ligand snapshots done in PB. (1)

_MMPBSA_complex_rism.out.# *rism3d.snglpnt* output file containing energy components of all complex snapshots done with 3D-RISM (1)

_MMPBSA_receptor_rism.out.# *rism3d.snglpnt* output file containing energy components of all receptor snapshots done with 3D-RISM (1)

_MMPBSA_ligand_rism.out.# rism3d.*snglpnt* output file containing energy components of all ligand snapshots done with 3D-RISM (1)

_MMPBSA_pbsanderoutput.junk.# File containing the information dumped by sander.APBS to STDOUT. (1)

_MMPBSA_ligand_nm.out.# Output file from *mmpbsa_py_nabnmode* that contains the entropy data for the ligand for all snapshots. (1)

_MMPBSA_receptor_nm.out.# Output file from *mmpbsa_py_nabnmode* that contains the entropy data for the receptor for all snapshots. (1)

_MMPBSA_complex_nm.out.# Output file from *mmpbsa_py_nabnmode* that contains the entropy data for the complex for all snapshots. (1)

_MMPBSA_mutant_... These files are analogs of the files that only start with _MMPBSA_ described above, but instead refer to the mutant system of alanine scanning calculations.

_MMPBSA_*out.# These files are thread-specific files. For serial simulations, only #=0 files are created. For parallel, #=0 through NUM_PROC - 1 are created.

30.3.7. Advanced Options

The default values for the various parameters as well as the inclusion of some variables over others in the general MMPBSA.py input file were chosen to cover the majority of all MM/PB(GB)SA calculations that would be attempted while maintaining maximum simplicity. However, there are situations in which MMPBSA.py may appear to be restrictive and ill-equipped to address. Attempts were made to maintain the simplicity described above while easily providing users with the ability to modify most aspects of the calculation easily and without editing the source code.

-make-mdins This flag will create all of the mdin and input files used by sander and nmode so that additional control can be granted to the user beyond the variables detailed in the input file section above. The files created are _MMPBSA_gb.mdin which controls GB calculation; _MMPBSA_pb.mdin which controls the PB calculation; _MMPBSA_sander_nm_min.mdin which controls the sander minimization of snapshots to be prepared for nmode calculations; and _MMPBSA_nmode.in which controls the nmode calculation. If no input file is specified, all files above are created with default values, and _MMPBSA_pb.mdin is created for AmberTools's *pbsa*. If you wish to create a file for sander.APBS, you must include an input file with "sander_apbs=1" specified to generate the desired input file. Note that if an input file is specified, only those mdin files pertinent to the calculation described therein will be created!

-use-mdins This flag will prevent MMPBSA.py from creating the input files that control the various calculations (_MMPBSA_gb.mdin, _MMPBSA_pb.mdin, _MMPBSA_sander_nm_min.mdin, and _MMPBSA_nmode.in). It will instead attempt to use existing input files (though they must have those names above!) in their place. In this way, the user has full control over the calculations performed, however care must be taken. The mdin files created by MMPBSA.py have been tested and are (generally) known to be consistent. Modifying certain variables (such as imin=5) may prevent the script from working, so this should only be done with care. It is recommended that users start with the existing mdin files (generated by the -make-mdins flag above), and add and/or modify parameters from there.

strip_mask This input variable allows users to control which atoms are stripped from the trajectory files associated with solvated_prmtop. In general, counterions and water molecules are stripped, and the complex is centered and imaged (so that if iwrap caused the ligand to "jump" to the other side of the periodic box, it is replaced inside the active site). If there is a specific metal ion that you wish to include in the calculation, you can prevent ptraj from stripping this ion by NOT specifying it in *strip_mask*. Note that *strip_mask* does nothing if no *solvated_prmtop* is provided.

QM/MMGBSA There are a lot of options for QM/MM calculations in *sander*, but not all of those options were made available via options in the MMPBSA.py input file. In order to take advantage of these other options, you'll have to make use of the -make-mdins and -use-mdins flags as detailed above and change the resulting _MMPBSA_gb_qmmm_com/rec/lig.mdin files to fit your desired calculation. Additionally, MMPBSA.py suffers all shortcomings of *sander*, one of those being that PB and QM/MM are incompatible. Therefore, only QM/MMGBSA is a valid option right now.

30.4. Python API

The aim of the MMPBSA.py API is to provide you with direct access to the raw data produced during a MMPBSA.py calculation. By default, MMPBSA.py calculates an average, standard deviation, and standard er-

Table 30.1.: *List and description of* `calc_key` *dict keys that may be present in instances of the* `mmpbsa_data` *class.*

Dictionary Key (`calc_key`)	Calculation Type
`'gb'`	Generalized Born Results
`'pb'`	Poisson-Boltzmann Results
`'rism gf'`	Gaussian Fluctuation 3D-RISM Results
`'rism std'`	Standard 3D-RISM Results
`'nmode'`	Normal Mode Analysis Results
`'qh'`	Quasi-harmonic Approximation Results

ror of the mean for all of the generated data sets, but does not support custom analyses. The API reads an `_MMPBSA_info` file, from which it will determine what kind of calculation you performed, then automatically parse the output files and load the data into arrays.

The `keep_files` variable in the `&general` section must be set to 1 or 2 in order to keep enough files for the API to work. It currently does NOT load decomposition data into available data structures. The topology files you used in the MMPBSA.py calculation must also be available in the location specified in the `_MMPBSA_info` file.

Using the API

The function `load_mmpbsa_info` takes the name of an MMPBSA.py info file (typically `_MMPBSA_info`) and returns a populated `mmpbsa_data` instance with all of the parsed data. An example code snippet that creates a `mmpbsa_data` instance from the information in `_MMPBSA_info` is shown below.

```
from MMPBSA_mods import API as MMPBSA_API
data = MMPBSA_API.load_mmpbsa_info("_MMPBSA_info")
```

Properties of mmpbsa_data

The `mmpbsa_data` class is a nested dictionary structure (`mmpbsa_data` is actually derived from `dict`). The various attributes of `mmpbsa_data` are described below followed by the defined operators.

Attributes

If the `numpy` package is installed and available, all data arrays will be `numpy.ndarray` instances. Otherwise, all data arrays will be `array.array` instances with the `'d'` data type specifier (for a double precision float). The data is organized in an `mmpbsa_data` instance in the following manner:

```
mmpbsa_data_instance[calc_key][system_component][energy_term]
```

In this example, `calc_key` is a `dict` key that is paired to another `dict` (`mmpbsa_data_instance` is the first-level `dict`, in this case). The keys of these second-level `dict` instances (`system_component`) pair to another `dict`. The keys of these inner-most (third-level) `dict` instances are paired with the data arrays for that energy term. The various dictionary keys are listed below for each level. If alanine scanning was performed, the `mmpbsa_data_instance` also has a “`mutant`” attribute that contains the same dictionary structure as `mmpbsa_data` does for the normal system. The only difference is that the data is accessed as follows:

```
mmpbsa_data_instance.mutant[calc_key][system_component][energy_term]
```

Note, all keys are case-sensitive, and if a space appears in the key, it must be present in your program. Also, if polar/non-polar decomposition is not performed for 3D-RISM, then the `'POLAR SOLV'` and `'APOLAR SOLV'` keys are replaced with the single key `'ERISM'`

Table 30.2.: *List and description of* `system_component` *keys that may be present in instances of the* `mmpbsa_data` *class.*

Dictionary Key (`system_component`)	Description
`'complex'`	Data sets for the complex. (Stability & Binding)
`'receptor'`	Data sets for the receptor. (Binding only)
`'ligand'`	Data sets for the ligand. (Binding only)

Table 30.3.: *List and description of* `energy_term` *keys that may be present in instances of the mmpbsa_data class. The allowed values of* `energy_term` *depend on the value of* `calc_key` *above in Table 30.1. The* `energy_term` *keys are listed for each calc_key enumerated above, accompanied by a description. The RISM keys are the same for both* `'rism gf'` *and* `'rism std'` *although the value of* `'POLAR SOLV'` *and* `'APOLAR SOLV'` *will differ depending on the method chosen. Those keys marked with * are specific to the CHARMM force field used through chamber. Those arrays are all 0 for normal Amber topology files.*

Description	`'gb'`	`'pb'`	`RISM`
Bond energy	`'BOND'`	`'BOND'`	`'BOND'`
Angle energy	`'ANGLE'`	`'ANGLE'`	`'ANGLE'`
Dihedral Energy	`'DIHED'`	`'DIHED'`	`'DIHED'`
Urey-Bradley*	`'UB'`	`'UB'`	—
Improper Dihedrals*	`'IMP'`	`'IMP'`	—
Correction Map*	`'CMAP'`	`'CMAP'`	—
1-4 van der Waals energy	`'1-4 VDW'`	`'1-4 VDW'`	`'1-4 VDW'`
1-4 Electrostatic energy	`'1-4 EEL'`	`'1-4 EEL'`	`'1-4 EEL'`
van der Waals energy	`'VDWAALS'`	`'VDWAALS'`	`'VDWAALS'`
Electrostatic energy	`'EEL'`	`'EEL'`	`'EEL'`
Polar solvation energy	`'EGB'`	`'EPB'`	`'POLAR SOLV'`
Non-polar solvation energy	`'ESURF'`	`'ENPOLAR'`	`'APOLAR SOLV'`
Total solvation free energy	`'G solv'`	`'G solv'`	`'G solv'`
Total gas phase free energy	`'G gas'`	`'G gas'`	`'G gas'`
Total energy	`'TOTAL'`	`'TOTAL'`	`'TOTAL'`

Table 30.4.: *Same as Table 30.3 for the entropy data.*

Description	`'nmode'`	`'qh'`
Translational entropy	`'Translational'`	`'Translational'`
Rotational entropy	`'Rotational'`	`'Rotational'`
Vibrational entropy	`'Vibrational'`	`'Vibrational'`
Total entropy	`'Total'`	`'Total'`

Defined operators

In-place addition: It extends all of the arrays that are common to both `mmpbsa_data` instances. This is useful if, for instance, you run two MMPBSA.py calculations, and you use `-prefix <new_prefix>` for the second simulation. Assuming that `<new_prefix>` is `_MMPBSA2_` for the second MMPBSA.py calculation, the following pseudo-code will generate an `mmpbsa_data` instance with all of the data in concatenated arrays. The pseudo-code assumes MMPBSA_mods.API was imported as demonstrated in Subsection 30.4.

```
data = MMPBSA_API.load_mmpbsa_info("_MMPBSA_info")
data += MMPBSA_API.load_mmpbsa_info("_MMPBSA2_info")
```

Example API Usage

In many cases, the autocorrelation function of the energy can aid in the analysis of MM/PBSA data, since it provides a way of determining the statistical independence of your data points. For example, 1000 correlated snapshots provide less information, and therefore less statistical certainty, than 1000 uncorrelated snapshots. The standard error of the mean calculation performed by MMPBSA.py assumes a completely uncorrelated set of snapshots, which means that it is a lower bound of the *true* standard error of the mean, and a plot of the autocorrelation function may help determine the actual value.

The example program below will calculate the autocorrelation function of the total energy (complex only for both the normal and alanine mutant systems) from a GB calculation and plot the resulting code using `matplotlib`.

```
import os
import sys
# append AMBERHOME/bin to sys.path
sys.path.append(os.path.join(os.getenv('AMBERHOME'), 'bin'))
# Now import the MMPBSA API
from MMPBSA_mods import API as MMPBSA_API
import matplotlib.pyplot as plt
import numpy as np

data = MMPBSA_API.load_mmpbsa_info('_MMPBSA_info')
total = data['gb']['complex']['TOTAL'].copy()

data = MMPBSA_API.load_mmpbsa_info('_MMPBSA_info')
total_mut = data.mutant['gb']['complex']['TOTAL'].copy()

# Create a second copy of the data set. The np.correlate function does not
# normalize the correlation function, so we modify total and total2 to get
# that effect
total -= total.mean()
total /= total.std()
total2 = total.copy() / len(total)
acor = np.correlate(total, total2, 'full')

total_mut -= total_mut.mean()
total_mut /= total_mut.std()
total2_mut = total_mut.copy() / len(total_mut)
acor_mut = np.correlate(total_mut, total2_mut, 'full')

# Now generate the 'lag' axis
xdata = np.arange(0, len(total))

# The acor data set is symmetric about the origin, so only accept the
# positive lag times. Graph the result
plt.plot(xdata, acor[len(acor)//2:], xdata, acor_mut[len(acor)//2:])
plt.show()
```

Decomposition Data

When performing decomposition analysis, the various decomp data is stored in a separate tree of dicts referenced with the '`decomp`' key. The key sequence is similar to the sequence for the 'normal' data described above, where `decomp` is followed by the solvent model (GB or PB), followed by the species (complex, receptor, or ligand), followed by the decomposition components (total, backbone, or sidechain), followed by the residue number (or residue pair for pairwise decomposition), finally followed by the contribution (internal, van der Waals, electrostatics, etc.) The available keys are shown in Figure 30.1 on page 707 (and each key is described afterwards).

Decomp Key Descriptions

gb All Generalized Born results

pb All Poisson-Boltzmann results

complex All results from the complex trajectory

receptor All results from the receptor trajectory

ligand All results from the ligand trajectory

TDC All results from the total decomposition

SDC All results from the sidechain decomposition

BDC All results from the backbone decomposition

All data from residue number "#" in per-residue decomposition (same residue numbering scheme as in each respective topology file)

#-## All interaction energies between residues "#" and "##" (same residue numbering scheme as in each respective topology file)

int Internal energy contributions (see the *idecomp* variable description above)

vdw van der Waals energy contributions

eel Electrostatic energy contributions

pol Polar solvation free energy contributions

sas Non-polar solvation free energy contributions

tot Total free energy contributions (sum of previous 5).

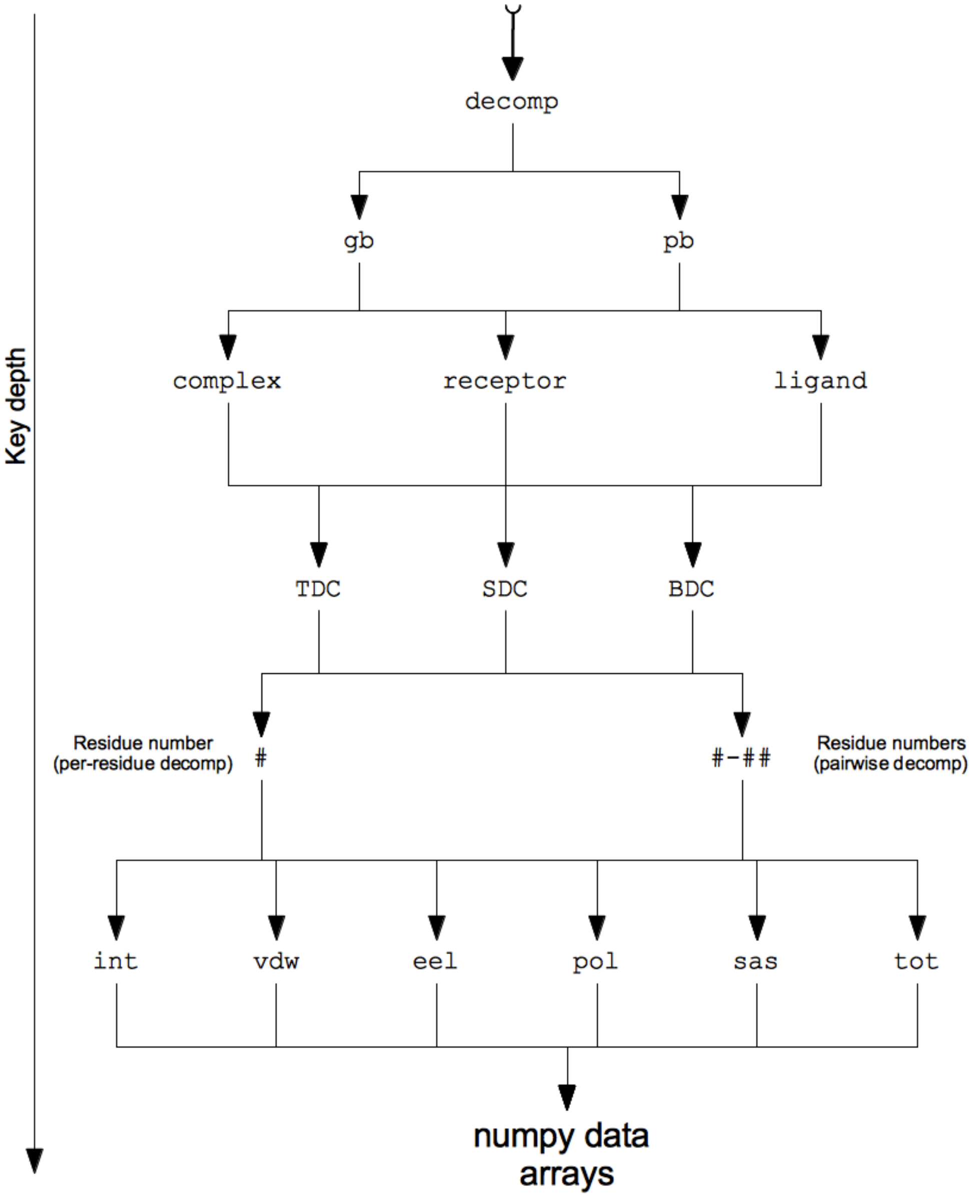

Key depth
decomp
gb
pb
complex
receptor
ligand
TDC
SDC
BDC
Residue number (per-residue decomp)
#
#-##
Residue numbers (pairwise decomp)
int
vdw
eel
pol
sas
tot
numpy data arrays

Figure 30.1.: *Tree of dict keys following the 'decomp' key in a* `mmpbsa_data` *instance.*

31. MM_PBSA

(*Note*: The recommended version of MM/PBSA to use is MMPBSA.py, described in Chapter 30.)

The MM_PBSA approach represents the postprocessing method to evaluate free energies of binding or to calculate absolute free energies of molecules in solution. The sets of structures are usually collected with molecular dynamics or Monte Carlo methods. However, the collections of structures should be stored in the format of an Amber trajectory file. The MM_PBSA/GBSA method combines the molecular mechanical energies with the continuum solvent approaches. The molecular mechanical energies are determined with the *sander* program from Amber and represent the internal energy (bond, angle and dihedral), and van der Waals and electrostatic interactions. An infinite cutoff for all interactions is used. The electrostatic contribution to the solvation free energy is calculated with a numerical solver for the Poisson-Boltzmann (PB) method, for example, as implemented in the *pbsa* program[165] or by generalized Born (GB) methods implemented in *sander*. Previous MM_PBSA applications were mostly performed with a numerical PB solver in the widely used *DelPhi* program,[168] which has been shown by Amber developers to be numerically consistent with the *pbsa* program. The nonpolar contribution to the solvation free energy has been determined with solvent-accessible-surface-area-dependent terms.[157] The surface area is computed with Paul Beroza's *molsurf* program, which is based on analytical ideas primarily developed by Mike Connolly.[567] An alternative method for nonpolar solvation energy is also included here. [173] The new method separates nonpolar contribution into two terms: the attractive (dispersion) and repulsive (cavity) interactions. Doing so significantly improves the correlation between the cavity free energies and solvent accessible surface areas for branched and cyclic organic molecules.[174] This is in contrast to the commonly used strategy that correlates total nonpolar solvation energies with solvent accessible surface areas, which only correlates well for linear aliphatic molecules.[157] In the new method, the attractive interaction is computed by a numerical integration over the solvent accessible surface area that accounts for solute solvent attractive interactions with an infinite cutoff.[175] Finally, estimates of conformational entropies can be made with the *nmode* or *NAB* module from Amber.

Although the basic ideas here have many precedents, the first application of this model in its present form was to the A- and B-forms of RNA and DNA, where many details of the basic method are given.[590] You may also wish to refer to reviews summarizing many of the applications of this model,[591, 592] as well as to papers describing more recent applications.[593–597]

The initial MM_PBSA scripts were written by Irina Massova. These were later modified and mostly turned into Perl scripts by Holger Gohlke, who also added GB/SA (generalized Born/surface area) options, and techniques to decompose energies into pairwise contributions from groups (where possible).

MM-PBSA should not be considered as a "black-box", and users should be familiar with Amber before attempting these sorts of calculations. These scripts automate a series of calculations, and cannot trap all the types of errors that might occur. ***You should be sure that you know how to carry out an MM-PBSA calculation "by hand" (i.e., without using the scripts);*** if you don't understand in detail what is going on, you will have no good reason to trust the results. Also, if something goes awry (and this is not all that uncommon), you will need to run and examine the individual steps to carry out useful debugging.

31.1. General instructions

The general procedure is to edit the *mm_pbsa.in* file (see below), and then to run the code as follows:

```
mm_pbsa.pl mm_pbsa.in > mm_pbsa.log
```

The *mm_pbsa.in* file refers to "receptor", "ligand" and "complex", but the chemical nature of these is up to the user, and these could equally well be referred to as "A", "B", and "AB". The procedure can also be used to estimate the free energy of a single species, and this is usually considered to be the "receptor".

The user also needs to prepare *prmtop* files for receptor, ligand, and complex using LEaP; if you are just doing "stability" calculations, only one of the *prmtop* files is required. Note that the *prmtop* files usually should only include the solutes, i.e., solvent molecules and counter ions should not be present. A sample LEaP command file for how to prepare *prmtop* files for solvated systems for the initial molecular dynamics runs and *prmtop* files for the systems in vacuum for the subsequent MM_PBSA calculations can be found in the *$AMBERHOME/src/mm_pbsa/Examples* directory.

The output files are labeled ".out", and the most useful summaries are in the "statistics.out" files. These give averages and standard deviations for various quantities, using the following labeling scheme:

```
*** Abbreviations for mm_pbsa output ***
ELE - non-bonded electrostatic energy + 1,4-electrostatic energy
VDW - non-bonded van der Waals energy + 1,4-van der Waals energy
INT - bond, angle, dihedral energies
GAS - ELE + VDW + INT
PBSUR - hydrophobic contrib. to solv. free energy for PB calculations
PBCAL - reaction field energy calculated by PB
PBSOL - PBSUR + PBCAL
PBELE - PBCAL + ELE
PBTOT - PBSOL + GAS
GBSUR - hydrophobic contrib. to solv. free energy for GB calculations
GB - reaction field energy calculated by GB
GBSOL - GBSUR + GB
GBELE - GB + ELE
GBTOT - GBSOL + GAS
TSTRA - translational entropy (as calculated by nmode) times temperature
TSROT - rotational entropy (as calculated by nmode) times temperature
TSVIB - vibrational entropy (as calculated by nmode) times temperature
*** Prefixes in front of abbreviations for energy decomposition ***
"T" - energy part due to _T_otal residue
"S" - energy part due to _S_idechain atoms
"B" - energy part due to _B_ackbone atoms
```

The *$AMBERHOME/src/mm_pbsa/Examples* directory shows examples of running a "Stability" calculation (*i.e.*, estimating the free energy of one species), a "Binding" calculation (estimating ΔG for A + B →AB), an "Nmode" calculation (to estimate entropies), and examples of how total energies (either by residue, or pair-wise by residue) and vibrational entropies (by residue only) can be decomposed. You should study the inputs and outputs in these directories to see how the program is typically used.

31.2. Input explanations

Below is a description of the input parameters for MM-PB/SA. A sample file can be found at *$AMBERHOME/src/mm_pbsa/Examples/mm_pbsa.in*. Further template files for variants of the implicit solvent models can be found in *$AMBERHOME/src/mm_pbsa/Examples/ TEMPLATE_INPUT_SCRIPTS*. There, the file *SURFTEN_SURFOFF_Recommendations.pdf* describes in detail recommended combinations of parameters for the implicit solvent models. The input file is structured into sections for different purposes. The parameters in the general section control which kind of operations are executed. Additional parameters for the chosen operations have to be defined in the later sections.

31.2.1. General

VERBOSE If set to 1, input and output files are not removed. This is useful for debugging purposes.

PARALLEL If set to values > 1, energy calculations for snapshots are done in parallel, using PARALLEL number of threads.

specifying snapshot location and naming

PREFIX	To the prefix of the snapshots, "{_com, _rec, _lig}.crd.Number" is added during generation of snapshots as well as during mm_pbsa calculations.
PATH	Specifies the location where to store or get snapshots.

selecting subsets of snapshots

START	Specifies the first snapshot to be used in energy calculations (optional, default is 1).
STOP	Specifies the last snapshot to be used in energy calculations (optional, default is 10e10).
OFFSET	Specifies the offset between snapshots in energy calculations (optional, default is 1). This may be interesting for entropy calculations via *nmode* or *NAB* to cut down on the number of snapshots to save computational time.

calculation of energy differences or absolute energies

COMPLEX	Set to 1 if free energy difference is calculated.
RECEPTOR	Set to 1 if either (absolute) free energy or free energy difference are calculated.
LIGAND	Set to 1 if free energy difference is calculated.

selection of parameter and topology files

COMPT	Parmtop file for the complex (not necessary for option GC).
RECPT	Parmtop file for the receptor (not necessary for option GC).
LIGPT	Parmtop file for the ligand (not necessary for option GC).

specification of operations/calculations

GC	Snapshots are generated from trajectories (see below).
AS	Residues are mutated during generation of snapshots from trajectories.
DC	Decompose the free energies into individual contributions (only works with MM, nmode, GB, and PB with the pbsa program of Amber).
MM	Calculation of gas phase energies using sander.
GB	Calculation of desolvation free energies using the analytical GB models in sander or numerical GB-NSR6 (see below).
PB	Calculation of desolvation free energies using delphi (see below). Calculation of nonpolar solvation free energies according to the NPOPT option in pbsa (see below).
MS	Calculation of nonpolar contributions to desolvation using molsurf (see below). If MS = 0 and GB = 1, nonpolar contributions are calculated with the LCPO (GBSA == 1) or the ICOSA (GBSA == 2) method in sander or with GBNSR6 (GBSA == 6). If MS = 0 and PB = 1, nonpolar contributions are calculated according the NPOPT option in pbsa (see below).
NM	Calculation of entropies with nmode or NAB.

31.2.2. Energy Decomposition Parameters

Energy decomposition is performed for gasphase energies, desolvation free energies calculated with GB or PB (using the pbsa program of Amber), nonpolar contributions to desolvation using the ICOSA method, and vibrational enrtopies using nmode. For amino acids and nucleotides, decomposition is also performed with respect to backbone and sidechain atoms. When doing a pairwise decomposition of the PB reaction field energy, one should note that for each included residue the PB equation has to be solved once per snapshot. Also a further decomposition into backbone and sidechain contributions has not been implemented for a pairwise PB decomposition.

specification of decomposition modus

DCTYPE	Values of 1 or 2 yield a decomposition on a per-residue basis. Values of 3 or 4 yield a decomposition on a pairwise basis. So far the number of pairs must not exceed the number of residues in the molecule considered. Values 1 or 3 add 1-4 interactions to bond contributions. Values 2 or 4 add 1-4 interactions to either electrostatic or vdW contributions.

residue assignment

COMREC	Residues belonging to the receptor molecule IN THE COMPLEX.
COMLIG	Residues belonging to the ligand molecule IN THE COMPLEX.
RECRES	Residues in the receptor molecule.
LIGRES	Residues in the ligand molecule.
{REC,LIG}MAP	Residues in the complex which are equivalent to the residues in the receptor molecule or the ligand molecule.

output filter

{COM,REC,LIG}PRI	Residues considered for output.

31.2.3. Poisson-Boltzmann Parameters

The following parameters are passed to the PB solver. Additional input parameters may also be added here. See the sander PB documentation for more options.

PROC	Determines which method is used for solving the PB equation. By default (PROC = 2) the pbsa program of the Amber suite is used.
REFE	Determines which reference state is taken for the PB calculation. By default (REFE = 0) reaction field energy is calculated with EXDI/INDI. Here, INDI must agree with DIELC from the MM section.
INDI	Dielectric constant for the solute.
EXDI	Dielectric constant for the surrounding solvent.
ISTRNG	Ionic strength (in mM) for the Poisson-Boltzmann solvent.
PRBRAD	Solvent probe radius in Angstrom: **1.4** with the radii in the prmtop files (default); **1.6** with the radii optimized by Tan and Luo (in preparation). See RADIOPT on how to choose a cavity radii set.

RADIOPT Option to set up radii for PB calc:

- **0** uses the radii from the prmtop file (default);
- **1** uses the radii optimized by Tan and Luo (in preparation) with respect to the reaction field energies computed in the TIP3P explicit solvents. Note that these optimized radii are based on Amber atom types (upper case) and charges. Radii from the ~.prmtop files are used if the atom types are defined by antechamber (lower case).

SCALE Lattice spacing in number of grids per Angstrom.

LINIT Number of iterations with the linear PB equation.

hybrid solvation model

IVCAP If set to 1, a solvent sphere (specified by CUTCAP, XCAP, YCAP, and ZCAP) is excised from a box of water.

If set to 5, a solvent shell is excised, specified by CUTCAP (the thickness of the shell in A). The electrostatic part of the solvation free energy is estimated from a linear response approximation using the explicit water plus a reaction field contribution from outside the sphere (i.e., a hybrid solvation approach is pursued).

In addition, the nonpolar contribution is estimated from a sum of (attractive) dispersion interactions calculated between the solute and the solvent molecules plus a (repulsive) cavity contribution. For the latter, the surface calculation must be done with MS = 1 and the PROBE should be set to 1.4 to get the solvent excluded surface.

CUTCAP Radius of the water sphere or thickness of the water shell. Note that the sphere must enclose the whole solute.

XCAP/YCAP/ZCAP Location of the center of the water sphere.

nonpolar solvation

Parameters for nonpolar solvation energies if MS = 0

INP Option for modeling nonpolar solvation free energy. See sander PB documentation for more information on the implementations by Tan and Luo (in preparation).

- **1:** uses the solvent-accessible-surface area to correlate total nonpolar solvation free energy: Gnp = SURFTEN * SASA + SURFOFF. Default.
- **2:** uses the solvent-accessible-surface area to correlate the repulsive (cavity) term only, and uses a surface-integration approach to compute the attractive (dispersion) term: Gnp = Gdisp + Gcavity = Gdisp + SURFTEN * SASA + SURFOFF. When this option is used, RADIOPT has to be set to 1, i.e. the radii set optimized by Tan and Luo to mimic Gnp in TIP3P explicit solvents. Otherwise, there is no guarantee that Gnp matches that in explicit solvents.

SURFTEN/SURFOFF Values used to compute the nonpolar solvation free energy Gnp acccording to INP. If INP = 1 and RADIOPT = 0 (default, see above), use SURFTEN/SURFOFF parameters that fit with the radii from the prmtop file, e.g., use SURFTEN: 0.00542; SURFOFF: 0.92 for PARSE radii. If INP = 2 and RADIOPT = 1, these two lines can be removed, i.e., use the default values set in pbsa for this nonpolar solvation model. Otherwise, set these to the following: SURFTEN: 0.04356; OFFSET: -1.008

Parameters for nonpolar solvation energies if MS = 1

SURFTEN/SURFOFF Values used to compute the nonpolar contribution Gnp to the desolvation according to either

(I) Gnp = SURFTEN * SASA + SURFOFF (if IVCAP = 0) or

(II) Gnp = Gdisp + Gcavity = Gdisp + SURFTEN * SASA + SURFOFF (if IVCAP > 0).

In the case of (I), use parameters that fit with the radii from the reaction field calculation. E.g., use SURFTEN: 0.00542, SURFOFF: 0.92 for PARSE radii or use SURFTEN: 0.005, SURFOFF: 0.86 for Tan & Luo radii. In the case of (II), use SURFTEN: 0.069; SURFOFF: 0.00 for calculating the Gcavity contribution.

31.2.4. Molecular Mechanics Parameters

The following parameters are passed to sander. For further details see the sander documentation.

DIELC	Dielectric constant for electrostatic interactions. Note: This is not related to GB calculations.

31.2.5. Generalized Born Parameters

IGB	Switches between Tsui's GB (1), Onufriev's GB (2, 5), Simmerling (8) and Onufriev's GBNSR6 (66).
GBSA	Switches between LCPO (1), ICOSA (2), and GBNSR6 (6) method for SASA calculation. Decomposition only works with ICOSA.
SALTCON	Concentration (in M) of 1-1 mobile counterions in solution. The salt concentration is internally converted to mM for GBNSR6.
EXTDIEL	Dielectric constant for the solvent.
INTDIEL	Dielectric constant for the solute.

SURFTEN/SURFOFF Values used to compute the nonpolar contribution Gnp to the desolvation free energy according to Gnp = SURFTEN * SASA + SURFOFF.

31.2.6. Molsurf Parameters

PROBE	Radius of the probe sphere used to calculate the SAS. In general, since Bondi radii are already augmented by 1.4A, PROBE should be 0.0 In IVCAP = 1 or 5, the solvent excluded surface is required for calculating the cavity contribution. Bondi radii are not augmented in this case and PROBE should be 1.4.

31.2.7. NMODE Parameters

The following parameters are passed either to NAB (for minimization and entropy calculation using gasphase statistical mechanics) or to sander (for minimization) and nmode (for entropy calculation using gasphase statistical mechanics). For further details see documentation.

PROC	Determines which method is used for the calculations: By default, PROC = 1, the NAB implementation of nmode is used. This allows using either a GB model or a distance-dependent dielectric for electrostatic energies. No entropy decomposition is possible, however. If PROC = 2, the "original" nmode implementation is used. Here, only a distance-dependent dielectric is available for electrostatic energies. Entropy decomposition is possible here, too.
MAXCYC	Maximum number of cycles of minimization.
DRMS	Convergence criterion for the energy gradient.
IGB	Switches between no GB (i.e., vacuum electrostatics) (0) or Tsui's GB (1).

SALTCON Concentration (in M) of 1-1 mobile counterions in solution.

EXTDIEL Dielectricity constant for the solvent.

SURFTEN Value used to compute the nonpolar contribution Gnp to the desolvation according to Gnp = SURFTEN * SASA.

DIELC (Distance-dependent) dielectric constant.

31.2.8. Parameters for Snapshot Generation

BOX "YES": means that periodic boundary conditions were used during MD simulation and that box information has been printed in the trajectory files; "NO": means opposite.

NTOTAL Total number of atoms per snapshot printed in the trajectory file (including water, ions, ...).

NSTART Start structure extraction from NSTART snapshot.

NSTOP Stop structure extraction at NSTOP snapshot.

NFREQ Every NFREQ structure will be extracted from the trajectory.

NUMBER_LIG_GROUPS Number of subsequent LSTART/LSTOP combinations to extract atoms belonging to the ligand.

LSTART Number of first ligand atom in the trajectory entry.

LSTOP Number of last ligand atom in the trajectory entry.

NUMBER_REC_GROUPS Number of subsequent RSTART/RSTOP combinations to extract atoms belonging to the receptor.

RSTART Number of first receptor atom in the trajectory entry.

RSTOP Number of last receptor atom in the trajectory entry.

Note: If only one molecular species is extracted, use only the receptor parameters (NUMBER_REC_GROUPS, RSTART, RSTOP).

31.2.9. Parameters for Alanine Scanning

The following parameters are additionally passed to make_crd_hg in conjunction with the ones from the snapshot generation section if "alanine scanning" is requested. The description of the parameters is taken from Irina Massova.

NUMBER_MUTANT_GROUPS Total number of mutated residues. For each mutated residue, the following four parameters must be given subsequently.

MUTANT_ATOM1 If residue is mutated to Ala then this is: a pointer on the CG atom of the mutated residue for all residues except Thr, Ile and Val; a pointer to CG2 if Thr, Ile or Val residue is mutated to Ala; or a pointer to OG if Ser residue is mutated to Ala. If residue is mutated to Gly then this is a pointer on CB.

MUTANT_ATOM2 If residue is mutated to Ala then this is: zero for all mutated residues except Thr, Val, and Ile; a pointer on OG1 if Thr residue is mutated to Ala; or a pointer on CG1 if Val or Ile residue is mutated to Ala. If residue is mutated to Gly then this should be always zero.

MUTANT_KEEP A pointer on the C atom (carbonyl atom) for the mutated residue.

MUTANT_REFERENCE If residue is mutated to Ala then this is a pointer on CB atom for the mutated residue. If residue is mutated to Gly then this is a pointer on CA atom for the mutated residue.

Note: The method will not work for a smaller residue mutation to a bigger for example Gly -> Ala mutation. Note: Maximum number of the simultaneously mutated residues is 40.

31.2.10. Trajectory Specification

The specified trajectories are used to extract snapshots with "make_crd_hg"

TRAJECTORY Each trajectory file name must be preceeded by the TRAJECTORY card. Subsequent trajectories are considered together. Trajectories may be in ascii as well as in .gz format. To be able to identify the title line, it must be identical in all files.

32. FEW

The Free Energy Workflow (FEW) is a tool for automated calculation of the binding free energy of a *set of ligands binding to the same receptor* using modules provided in the AMBER suite of programs. Prerequisite for calculations with FEW is the existence of 3D complex structures of a receptor and ligands. Generally, the more accurate the complex structures are the more accurate results can be expected.

FEW provides functions for setup of three types of binding free energy calculations: implicit solvent calculations by the MM-PBSA or MM-GBSA methods, linear interaction energy analyses (LIE), and thermodynamic integration (TI) calculations. These three binding free energy calculation approaches are available via three program modules provided in FEW:

- WAMM: Workflow for automated MM-PBSA & MM-GBSA
- LIEW: Linear interaction energy workflow
- TIW: Thermodynamic integration workflow

32.1. Installation

The program FEW consists of the main Perl script "FEW.pl" and a set of Perl modules stored in the folder "libs" provided in the main FEW directory.

A perl installation (version 5.10 or newer) needs to be available on the system where FEW shall be executed. For running the program some additional Perl modules are needed (Table 32.1), which are provided under the terms of the respective license in the folder "additional_libs". Please ensure that the "additional_libs" folder is located in the same directory in which the FEW.pl script resides.

FEW can be used with Amber and AmberTools. To enable access of the program FEW to AmberTools, the tools need to be executable on the system by just calling their names, e.g., "antechamber" should invoke the *antechamber* program. The following tools and programs are used by FEW directly: *ambpdb*, *tleap*, *antechamber*, *cpptraj*, *parmchk*, *mm_pbsa.pl and Babel* [598] (in case SDF-input files are provided). In addition, the AMBER programs *sander* and/or *PMEMD* are required, and if charges shall be calculated by the RESP procedure also access to the program *Gaussian03* is needed. The later programs can be installed on a different system or a compute cluster.

Table 32.1.: *Perl modules from CPAN used by FEW.*

Module name(1)	Functionality
PerlMol	Read and manage atom information
FreezeThaw	Interconversion between Perl structures and strings
File::ReadBackwards	Read file line by line from end of file
Statistics::Normality Statistics::PointEstimation Statistics::Descriptive Statistics::Smoother Statistics::Distributions	Modules for statistical analysis

(1) Modules are provided with FEW under the terms of the respective license.

Basic program call

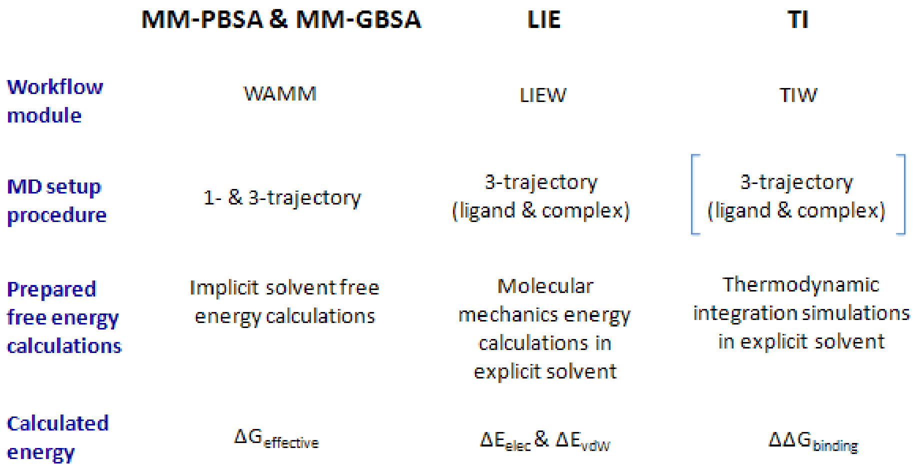
perl FEW.pl <procedure> <command-file>

Table 32.2.: *Overview of procedures and corresponding modules available in FEW.*

Procedure name	Program module used	Key phrase in command file[1]
MMPBSA or MMGBSA[2]	WAMM	@WAMM
LIE	LIEW	@LIEW
TI	TIW	@TIW

[1] Expression that needs to be provided in the first line of the command file to ensure that the requested procedure and the provided command file match.

[2] Either MMPBSA or MMGBSA can be specified.



	MM-PBSA & MM-GBSA	LIE	TI
Workflow module	WAMM	LIEW	TIW
MD setup procedure	1- & 3-trajectory	3-trajectory (ligand & complex)	[3-trajectory (ligand & complex)]
Prepared free energy calculations	Implicit solvent free energy calculations	Molecular mechanics energy calculations in explicit solvent	Thermodynamic integration simulations in explicit solvent
Calculated energy	$\Delta G_{effective}$	ΔE_{elec} & ΔE_{vdW}	$\Delta\Delta G_{binding}$

Figure 32.1.: *Overview of program modules and functionality provided in FEW. All three free energy calculation workflows available in FEW have a MD setup step in common.*

The procedures that can be chosen are listed in Table 32.2, and an overview of the functionality provided in the individual free energy calculation modules is shown in Figure 32.1. Example command files can be found in the folder `$AMBERHOME/AmberTools/src/FEW/examples/command_files`. Please ensure that in each command file the program module that shall be used for calculation is specified via a key phrase in the first line (Table 32.2).

In addition, template files, e.g., input files with parameters for MD simulations, are available under `examples/input_info`. It is strongly recommended that non-experts use these template files for analysis and make only those system and/or computing resource specific modifications that are requested below.

A complete example analysis corresponding to the show case example presented in ref. [599] including all input files for setup and the final result files with the computed binding free energies can be obtained from `http://cpclab.uni-duesseldorf.de/software`. The current version of FEW uses per default the ff12SB force field of AMBER. Earlier FEW versions, as the one used for the generation of the case study data, employed the ff99SB force field. For backwards compatibility with previous FEW versions set the flag `backwards` to `1`.

32.2. Overview of workflow steps and minimal input

A detailed description of FEW and its functionality is provided in ref. [599]. We strongly encourage the user to run the FEW tutorial first that is available at `http://ambermd.org/tutorials`.

For the setup of free energy calculations with FEW a 3D receptor structure in PDB format and 3D ligand structures with coordinates of the ligand bound position in mol2 format are required (see section 32.3.1). FEW provides besides the general setup functionality a lot of additional system / computing architecture specific and expert options that can be requested by setting parameters / flags in the command file. All available options are described in the following sections, where essential parameters are marked in bold, while optional additional parameters are shown in normal writing. For a typical system it is usually sufficient to define the essential flags. Example files containing only those flags that are commonly needed can be found under `$AMBERHOME/AmberTools/src/FEW/examples/command_files/minimalistic_files`. Please use these files only if your ligands are available as single structure mol2 files and if the receptor contains only standard residues defined in the ff12SB force field.

The setup of free energy calculations with FEW is conducted in a multi-step procedure, i.e., FEW is called several times using a command file with the parameters for the respective setup step. The Table 32.3 shows the minimum number of FEW calls required for preparation and analysis of the individual free energy calculations. The individual setup steps can be further devided into individual tasks, such that each setup task can be tracked and checked. The later is generally recommended if any problems are encountered in the setup procedure. In this case it should also be thoroughly checked, whether additional parameters might need to be specified for the the specific system. Example command files of the individual setup steps of the different setup procedues containing all available parameters can be found in the procedure specific folders under `$AMBERHOME/AmberTools/src/FEW/examples/command_files`

Table 32.3.: *Overview of steps required for setup, execution, and analysis of MM-PB(GB)SA, LIE, and TI calculations with FEW* 1).

Call 2)	**MM-PB(GB)SA** (Section **32.4**)	**LIE** (Section **32.5**)
	MD simulations (Section **32.3**)	
	RESP charges	**AM1-BCC charges**
X	Preparation of Gaussian input files (32.3.2)	Charge calculation & setup of MD simulations (32.3.2)
	Calculation of ESP with Gaussian	
X	Charge calculation & setup of MD simulations (32.3.2)	
	Running MD simulations	
	Free energy calculations	
X	Setup of MM-PB(GB)SA calculations (32.4)	Setup of LIE analysis (32.5)
	Running MM-PB(GB)SA calculations	*Running LIE calculations*
	Preparation of MM-PB(GB)SA results for analysis (32.4)	Preparation of LIE results for analysis (32.5)

32.3. Common setup of molecular dynamics simulations

The setup of molecular dynamics (MD) simulations with FEW can be used in connection with all three available free energy calculation procedures (cf. Figure 32.1).

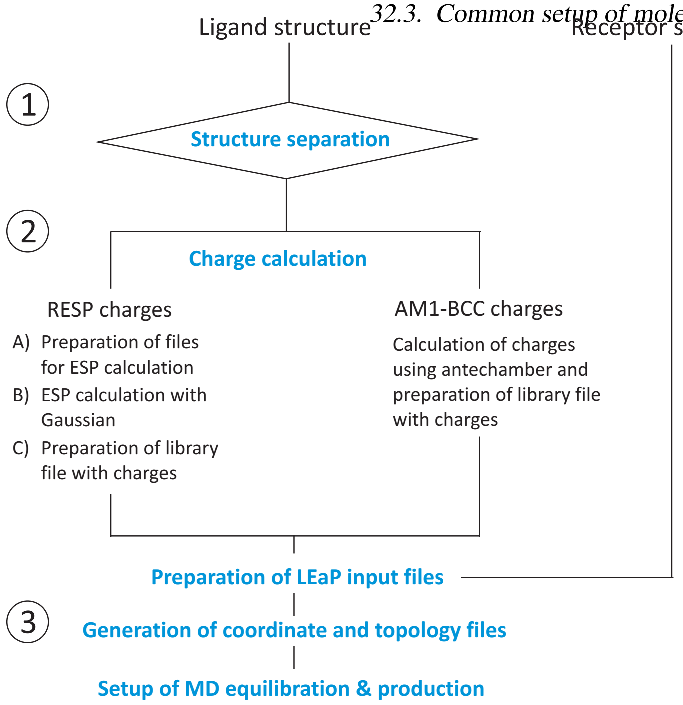

Ligand structure
Receptor structure
1
Structure separation
2
Charge calculation
RESP charges
A) Preparation of files for ESP calculation
B) ESP calculation with Gaussian
C) Preparation of library file with charges
AM1-BCC charges
Calculation of charges using antechamber and preparation of library file with charges
Preparation of LEaP input files
3
Generation of coordinate and topology files
Setup of MD equilibration & production

Figure 32.2.: *Graphical illustration of the steps conducted for setup of MD simulations. Steps that can be executed independently by separate calls of FEW.pl are indicated by numbers. However, the individual steps can be combined, so that the whole MD simulation setup can be conducted in 1 or 2 steps for the AM1-BCC and RESP charge option, respectively (see 32.3).*

MM-PB/GBSA and LIE calculations require the existence of MD trajectories from which snapshots can be extracted, so that a MD setup is needed. For TI calculations it is recommended to use structures pre-equilibrated with the common MD setup functionality of FEW as input structures. Expert users may also provide structures directly, i.e. without using the MD equilibration preparation functionality of FEW. In the later case the structures for TI input must be prepocessed using the structure preparation workflow available in the MD setup procedure (see Section 32.3.2). As the MD setup functionality requires the same input for all three procedures, it is discussed here separately from the procedure specific features. The setup of MD simulations is conducted in 3 consecutive steps (see Figure 32.2), which can be initiated by a minimum of 1 or 2 FEW calls in the case of a setup of MD simulations with AM1-BCC charges or RESP charges for the ligands, respectively (cf. Table 32.3).

32.3.1. Input structures

Ligand structures: 3D coordinates of ligand structures in the bound position and with the correct protonation state need to be provided in one of the following formats:

- A) SDF file containing multiple ligand structures (requires the program Babel [598])
- B) mol2 file with multiple ligand structures
- C) mol2 files with one structure per file

In the case of A) and B) a structure separation needs to be requested using the flag `structure_separation` in the command file. This will result in a set of structures in format C), which is required for MD setup and all further calculations. Ligands must consist of no more than one residue, and mol2 files must obey the formatting rules defined by TRIPOS (see http://www.tripos.com/data/support/mol2.pdf). In addition to the information obligatory according to these rules for the entries in the ATOM section of mol2 files, FEW requires the substructure ID and the substructure name, i.e., the residue ID and name. As residue names will be shortend to three characters, it is recommended to use ligand residue names that consist of three characters only. Residue names can consist of letters and numbers, but should not start with a number nor contain special characters.

Receptor structure: A structure of the receptor in PDB format with all atoms that shall be considered in the calculation, i.e., including protons, is required. This structure can contain crystal water and / or non-standard residues. The residues of the receptor need to be consecutively numbered starting from residue number 1. To ensure that the atom names of the PDB structure can be recognized by LEaP, it is recommended to load the prepared PDB file first into LEaP and then re-save it. By this the residues are also automatically re-numbered according to the requirements of FEW. If there are different chains or missing residues in the receptor structure, those parts of the structure that are not directly connected need to be separated by a TER card in the PDB file (see http://deposit.rcsb.org/adit/docs/pdb_atom_format.html). The residue name of all atoms that belong to water molecules must be either “WAT” or “HOH”.

32.3.2. Flags for MD setup

The following flags are available for MD setup. Flags and corresponding options are given. Essential flags are marked in bold and optional ones are shown in normal writing. Statements in "<" and ">" brackets denote place holders. For example input files see `$AMBERHOME/AmberTools/src/FEW/examples/command_files/commonMDsetup`. MD simulations are setup with a cubic water box extending at least 11 Å in each direction from the solute. Truncated octahedrons are currently not supported. The normal file extensions of MD input and output files are shortened: *.inpcrd to *.crd and *.prmtop to *.top. An overview of the folder structure created upon MD setup is shown in Figure 32.3.

Specification of input / output directories and formats:

lig_struct_path <path>	Path to folder containing the ligand structures. For ligands provided in format C) a folder containing exclusively all ligand structures that shall be regarded needs to be manually created and specified under `lig_struct_path`. If ligand structures are provided in input format A) or B) and a separation is requested a folder called `structs` containing the separated structures is created in the basic output directory. This folder needs to be specified in all subsequent setup steps.
output_path <path>	Path to main output directory in which all new folders will be generated.
rec_structure <structure>	Full path and name of receptor structure file in PDB format.
lig_format_sdf 0 | 1	Set to 1, if multi-ligand file in sdf-format is provided; format A).
lig_format_mol2 0 | 1	Set to 1, if ligand structure files are provided in format B) or C).
water_in_rec 0 | 1	Optional: 1: Crystal water present in receptor structure. Water molecules need to be provided after the solute and should carry the residue name "WAT" or “HOH”. 0: PDB structure of the receptor contains only the solute and no crystal water molecules.
multi_structure_lig_file <name>	Only relevant for ligands in input formats A) or B): Basic name of ligand input file if multi-structure file is provided in input formats A) or B). File extension can be omitted.
bound_rec_structure <structure>	Optional: Absolute path and name of the receptor PDB structure in the bound state, in case two different receptor structures shall be used for setup of complex and receptor in the 3-trajectory approach.
membrane_file <structure>	Optional: Absolute path and name of a PDB file containing lipids, ions, and water molecules. This file is only required if a MD simulation with an explicit

membrane shall be performed. The file needs to be generated using external tools, e.g. the CHARMM-GUI Membrane Builder (http://www.charmm-gui.org/?doc=input/membrane) [600–603]. It is recommended to use the latter tool for preparing a PDB file of the membrane, water, and ions, if the Lipid14 force field [70] shall be used for the MD simulations. The files generated with the CHARMM-GUI Membrane Builder can be converted with the `charmmlipid2amber.py` script provided with AMBER in order to obtain the required Lipid14 specific lipid naming scheme. If the file containing lipids, ions, and water is generated with another program, the user needs to ensure that the file formatting and lipid naming scheme is consistent with AMBER and the force fields that shall be used.

Structure separation

structure_separation 0 | 1	Only relevant if ligands are in input formats A) or B): Set to 1 in case of ligand input format A) or B). If set to 1, structure separation is conducted, and the resulting single structure files are stored in mol2 format in a folder called `structs` under `<output_path>`. Default = 0.

Generation of files for setup of system with *LEaP*

prepare_leap_input 0 | 1	The parameters in this section will only be regarded if this flag is set to 1. If the flag is switched on, the files needed for the preparation of the system with LEaP are generated.
non_neutral_ligands 0 | 1	Set to 1, if the total charge of at least one ligand molecule is not equal to zero. In this case the total charge of each non-neutral ligand molecule needs to be defined in a separate file `lig_charge_file`.
lig_charge_file <file>	If the total charge of at least one ligand molecule is not equal to zero, specify the full path and name of a file in which the names, the total charge, and the multiplicity of the non-neutral ligands is stored in tab-separated format; see `examples/input_info/charge.txt`.
am1_lig_charges 0 | 1	Set to 1 if ligand charges shall be calculated according to the AM1-BCC method [329, 330]. Please note: Only one charge calculation method can be used at a time.
resp_lig_charges 0 | 1	Set to 1 if ligand charges shall be calculated according to the "Restraint electrostatic potential fit" (RESP) method [331]. Please note: Only one charge calculation method can be used at a time.
resp_setup_step1 0 | 1	Request step one of the RESP charge calculation. The RESP charges are calculated in two steps. First, the files needed for ligand structure optimization and the calculation of the electrostatic potential with the program *Gaussian* are generated. If this step is carried out, a folder called “gauss” containing all input files for the *Gaussian* calculation is generated in the `<output_path>` directory. This folder can be copied to a compute cluster, where the program *Gaussian* is available. It is then possible to run the *Gaussian* jobs for all ligands at the same time.
resp_setup_step2 0 | 1	Request step two of the RESP charge calculation, in which the atomic charges are calculated based on the ESP computed with *Gaussian*. If this flag is set to 1, the *Gaussian* output files need to be available in the folder `<output_path>/gauss`.
gauss_batch_file 0 | 1	Optional: Request setup of batch scripts for *Gaussian* jobs. Default = 0.

gauss_batch_template <file>	In case `resp_lig_charges=1`, `resp_setup_step1=1`, and `gauss_batch_file=1`, then the full path and name of the template file for the generation of the *Gaussian* batch-script needs to be specified here. Example template file: `examples/input_info/gaussian.pbs`. Please adapt the file according to the needs of your queuing system, but keep the variables and the format in the section "Fix variables" and ensure that the line for job naming ends with "-N".
gauss_batch_path <path>	If the basis working directory for the *Gaussian* calculations differs from the `<output_path>` directory the new basis directory can be specified here. For example, this might be the case if the calculations shall be run on a compute cluster.
average_charges <file>	Optional: If the charges of two enantiomers shall be averaged, such that the two molecules obtain the same atomic charges, a file in which the enantiomer pairs are defined needs to be specified here. Prerequisite: The atom order and naming in the input mol2-files of the ligand isomers is identical. An example file can be found under `examples/input_info/isomer_pairs.txt`
calc_charges 0 | 1	Optional: This flag determines whether charges are calculated. If set to 0, only LEaP input files that do not require charge calculation are generated. Default = 1.
prepare_membrane 0 | 1	Optional: Request setup of MD simulation with explicit membrane. Only if `prepare_membrane=1` the lipids, ions, and water molecules specified in the `membrane_file` will be considered. Default = 0.
ligand_water_cutoff <no.>	Optional: Relevant only if `prepare_membrane=1`. Cutoff distance from the ligand within which all water molecules will be removed upon ligand insertion in order to avoid clashes between the ligand and water molecules. Default = 1 Å.

Setup of MD simulations

setup_MDsimulations 0 | 1	Request generation of input files for MD simulations by setting this flag to 1. All other flags in this section are only taken into account if `setup_MDsimulations=1`.
traj_setup_method 1 | 3	Specify whether simulations shall be setup according to the 1-trajectory or the 3-trajectory protocol for MM-PBSA or MM-GBSA. For LIE analyses, only the 3-trajectory setup, i.e., separate simulations for the ligand bound to the complex and for the ligand free in solution, works. For the TI approach preparation of an equilibration according to the 3-trajectory setup can be performed.
MD_am1 0 | 1	Set to 1 if MD simulation setup shall be conducted using previously calculated AM1-BCC charges.
MD_resp 0 | 1	Set to 1 if setup of MD simulations shall be carried out using previously calculated RESP charges.
SSbond_file <file>	If your receptor contains disulfide bridges the S-S bond connectivities need to be defined in a separate file. The full path and name of the file containing the disulfide bridge definitions should be provided here. In this file the numbers of those residues involved in S-S bonds should be specified in tab-separated format. Please note, all cysteine residues involved in S-S bonds should be named CYX in the provided receptor PDB structure. For an example S-S connectivity file see `examples/input_info/SSbridges.txt`

total_MDequil_time <time>	Total equilibration time in [ps]. The simulation time requested in all template files provided for equilibration needs to sum up to the time provided here. In case the files provided in the example `MDequil_template_folder` are used this keyword does not need to be specified. Default = 400 ps.
MDequil_batch_template <file>	Absolute path and name of the batch template file for the equilibration. This file should contain calls for all equilibration steps. For an example template file see `examples/input_info/equi.pbs`. Please adapt this file according to your needs, but keep the variables and the format in the section "Fix variables" and ensure that the line for job naming ends with "-N".
total_MDprod_time <time>	Total simulation time of MD production in [ns].
MD_prod_batch_template <file>	Absolute path and name of the batch template file for MD production. For an example template file see `examples/input_info/prod.pbs`. Please adapt this file according to the needs of your queuing system, but do not change anything from the section "Fix variables" up to the section "Requeue" and ensure that the line for job naming ends with "-N".
no_of_rec_residues <no.>	Actual number of residues in the receptor structure when all residues in the receptor are consecutively numbered starting from 1. Structurally bound ions should be treated as part of the receptor.
restart_file_for_MDprod <file>	Basename of restart file from equilibration that shall be used as initial file for MD production.
additional_library <library file>	Absolute path and name of additional library file. If your receptor structure contains non-standard residues or ions, an AMBER library file for these residues / ions needs to be provided here.
additional_frcmod <file>	Absolute path and name of additional parameter file. If your receptor structure contains residues or ions for which no parameters are available in the ff12SB force field, a parameter file in which the missing parameters are defined needs to be provided here.
MD_batch_path <path>	If the simulations need to be conducted on another system / machine than the one used for setup, the `<output_path>` during the simulations may differ from the one used for setup. If this is the case, please specify here the basis directory for the MD simulations. If no path is defined, it is assumed that the path is equal to `<output_path>`.
MDequil_template_folder <folder>	Absolute path to the folder containing the template files for equilibration. All files provided in this folder will be considered for equilibration setup. Example equilibration files that will be used per default can be found under `examples/input_info/equi`. If you change the template files or create additional files, please keep the format for the definition of the residues that shall be restraint.
MDprod_template <file>	Please specify the absolute path and name of the template file for production run. In this file all the flags you would like to use in your MD simulation should be set according to the *sander* | *PMEMD* definitions. The assignment should have the form flag = <value>, and individual flags should be separated by commas. For an example file see `examples/input_info/MD_prod.in`. Per default this file will be used as template if no template file is specified.

water_model TIP3P | OPC	Water model that shall be used for the MD simulations. Currently the water models TIP3P and OPC are available.

Additional parameters for setup of MD simulations with explicit membrane

prepare_membrane 0 | 1	Optional: Request setup of MD simulation with explicit membrane. Default = 0.
use_lipid14_ff 0 | 1	If set to `1`, the Lipid14 force field will be used for the lipids in the explicit membrane simulation. In case a setup of a MD simulation with explicit membrane is requested (`prepare_membrane=1`) although `use_lipid14_ff` is not specified or `use_lipid14_ff=0` and `use_gaff_lipid_ff=0`, then `use_lipid14_ff` is set to `1` per default.
use_gaff_lipid_ff 0 | 1	If `use_gaff_lipid_ff=1` parameters from the GaffLipid force field will be used. Please note that in this case library and parameter files for the lipids need to be provided under `additional_library` and `additional_frcmod` file (see above). These files can be obtained from the Lipidbook repository at http://lipidbook.bioch.ox.ac.uk [604].
restrain_membrane_residues <no.>	Membrane residues that shall be restrained during the equilibration phase of the MD simulations. This parameter needs only to be provided if `prepare_membrane=1`. Attention: The number of membrane residues differs from the number of lipids if the Lipid14 force field is used. In this case usually residue number = 3 × lipid number. Default = 0.

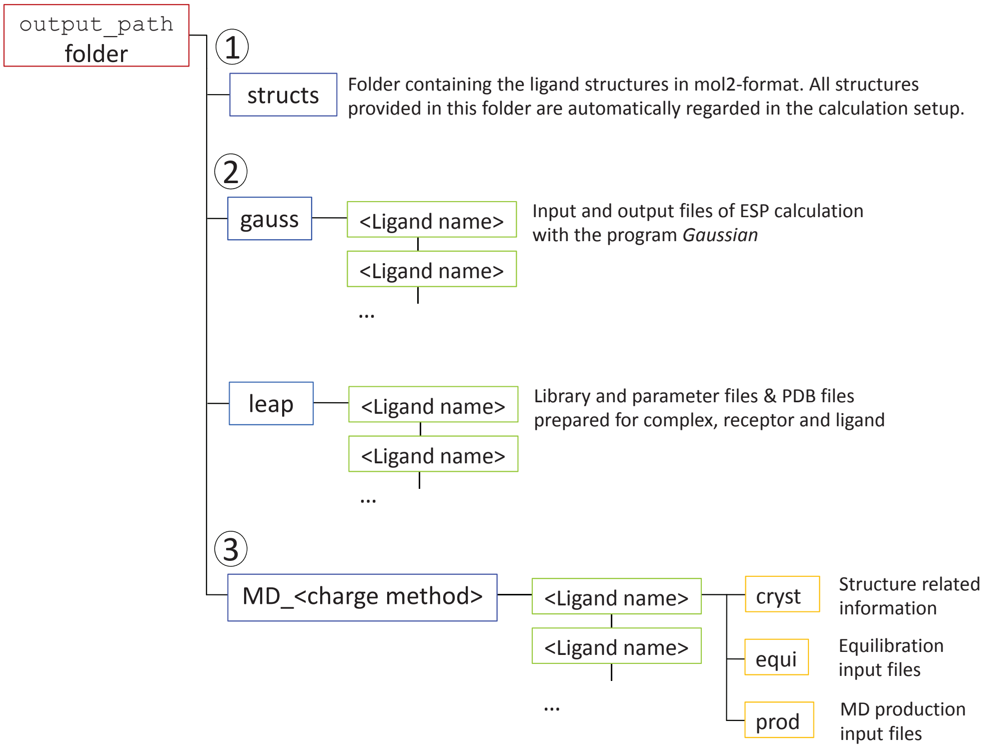

output_path folder
①
structs
Folder containing the ligand structures in mol2-format. All structures provided in this folder are automatically regarded in the calculation setup.
②
gauss
<Ligand name>
<Ligand name>
...
Input and output files of ESP calculation with the program *Gaussian*
leap
<Ligand name>
<Ligand name>
...
Library and parameter files & PDB files prepared for complex, receptor and ligand
③
MD_<charge method>
<Ligand name>
<Ligand name>
...
cryst
Structure related information
equi
Equilibration input files
prod
MD production input files

Figure 32.3.: *Directory structure and files created during the common MD setup step of FEW.*

32.4. Workflow for automated MM-PBSA & MM-GBSA calculations (WAMM)

The module WAMM allows to calculate binding free energies of ligands according to four flavors of the Molecular Mechanics Poisson-Boltzmann Surface Area (MM-PBSA) approach and three types of the Molecular Mechanics Generalized Born Surface Area (MM-GBSA) approach. All energies are calculated based on conformational ensembles generated by MD simulations that have been conducted using the common MD setup functionality of FEW (Section 32.3). An overview of the available binding free energy calculation options is given in Table 32.4. All binding free energy calculation methods except method PB2 can be applied to the 1- and the 3-trajectory approach. PB2 can only be used in conjunction with the 3-trajectory approach. Residue-wise and pair-wise decomposition of the effective energy (`decomposition` keyword) is currently only possible with PB=4 & GB=1. The solvent accessible surface area is calculated according to the ICOSA method in this case.

The availability of trajectories from MD productions for the complex (1-trajectory approach) or for complex, receptor, and ligand (3-trajectory approach) is prerequisite for the setup of free energy calculations according to the MM-PBSA / MM-GBSA method. These trajectories should be prepared with the MD setup functionality of FEW (cf. section 32.3). Example files for WAMM analysis setup can be found in `$AMBERHOME/AmberTools/src/FEW/examples/command_files/MMPBSA`. Besides the common section for input / output directories and format definitions, the WAMM module considers several specific flags (see below). An overview of the folder structure created by the MM-PB(GB)SA workfow is shown in Figure 32.4.

Table 32.4.: *Overview of flavors of MM-PBSA and MM-GBSA calculation procedures available in the WAMM module.*

Alias	Radii 1)	Calculation of polar solvation energy	Method for calculation of the non-polar solvation energy			
			SASA 2)	$E_{nonpolar}$ 3)	γ 4)	b 5)
GB1	mbondi [154]	GB^{HTC} [141, 152, 154]	LCPO [126]	γ SASA + b	0.00720	0.0000
GB2	mbondi2 [138]	GB^{OBC}model I [138]	LCPO [126]	γ SASA + b	0.00500	0.0000
GB5	mbondi2 [138]	GB^{OBC}model II [138]	LCPO [126]	γ SASA + b	0.00500	0.0000
GB6	bondi	GBNSR6 [605], 5.1	GBNSR6	γ SASA + b	0.00500	0.0000
PB1	Tan&Luo + mbondi 6) [154, 187]	PBSA 7)	PBSA 8) [173]	γ SASA + b + $E_{dispersion}$.	0.03780	- 0.5692
PB2	Tan&Luo + mbondi 6) [154, 187]	Hybrid PBSA 9) [606]	Molsurf [567] + PBSA	γ MSA + b + E_{vdW} 10)	0.06900	0.0000
PB3	Parse [157]	PBSA 7)	Molsurf [567]	γ SASA + b	0.00542	0.9200
PB4	mbondi [154]	PBSA 7)	Molsurf [567]	γ SASA + b	0.00720	0.0000
dec 11)	mbondi [154]	PBSA 7)+ GB^{HTC} [141, 152, 154]	ICOSA 12)	γ SASA + b	0.00720	0.0000

1) Radii used for the calculation of the polar solvation free energy.
2) Program or method used for the calculation of the solvent accessible surface area
3) Equation used for the calculation of the nonpolar part of the solvation free energy
4) Surface tension (SURFTEN) term in MM-PBSA / MM-GBSA calculations
5) Offset (SURFOFF) term in MM-PBSA / MM-GBSA calculations
6) Tan&Luo radii for the protein and mbondi radii for the ligand (per default). Radii optimized according to Tan&Luo [187] can be provided in the topology file and will then be regarded in the calculation setup.
7) Calculations are conducted with the PBSA module using the “Modified Incomplete Choleski Conjugate Gradient” Poisson-Boltzmann solver.
8) $E_{dispersion}$ is calculated by a numerical determination of the solvent accessible surface area.
9) Hybrid solvent MM-PBSA calculation according to Metz & Gohlke 2006. Please refer to the respective mm_pbsa.pl execution example provided in Amber 14 for a detailed explanation of the results and their correct interpretation.
10) The nonpolar solvation free energy is calculated as the sum of the cavity free energy γ MSA + b (where MSA = molecular surface area) and the van der Waals interaction energy between solute and solvent atoms.
11) Decomposition of effective binding free energies requested by the `decomposition` option.
12) SASA is calculated by a recursive approximation of a sphere around an atom, starting from an icosahedron.

Specification of input / output directories and formats

lig_struct_path <path> Path to folder containing the ligand structures. All ligand structures should now be available in mol2 format, since the conversion should have been carried out in the MD simulation preparation step.

output_path <path> Path to the basic output directory. This path should be identical to the `<output_path>` specified in the common MD setup step.

water_in_rec 0 | 1 Set to 1 if crystal water molecules were present in the receptor structure used for MD simulation setup.

General parameters for MM-PBSA / MM-GBSA calculation setup

mmpbsa_calc 0 | 1 Request setup of files for MM-PBSA or MM-GBSA calculations.

1_or_3_traj 1 | 3	Specification of the method that shall be used for calculation setup. *1-trajectory approach:* Requires complex trajectories prepared using `traj_setup_method=1` in the MD setup step. *3-trajectory approach:* Requires trajectories of ligand, receptor, and complex prepared with `traj_setup_method=3` in the MD setup step.
charge_method am1 | resp	Charge method that shall be used for the calculations. MD trajectories in which the corresponding charge method was employed for the ligand need to be available. See section 32.3 on how to setup the MD simulations.
additional_library <file>	Optional: Absolute path and name of additional library file. Such a library file is only required if the receptor structure contains non-standard residues.
additional_frcmod <file>	Optional: Absolute path and name of additional parameter file. Such a file is only needed, if not all parameters required to describe the receptor are available in the ff12SB force field.
mmpbsa_pl <file>	Absolute path and name of mm_pbsa.pl executable that shall be used for the calculations. Also a path relative to the AMBERHOME directory can be specified. Note that in the latter case the AMBERHOME variable needs to be set in the `mmpbsa_batch_template` batch template script. Per default it is assumed that mm_pbsa.pl can be called by `$AMBERHOME/bin/mm_pbsa.pl`

Snapshot extraction

extract_snapshots 0 | 1	Request coordinate extraction.
first_snapshot <number>	Number of the first structure that shall be extracted. Please consider that <number> is equivalent to the sum of the number of the structure in the corresponding trajectory and the number of structures present in all trajectories read in before.
last_snapshot <number>	Number of last structure that shall be extracted. Please consider that <number> is equivalent to the sum of the number of the structure in the corresponding trajectory and the number of structures present in all trajectories read in before.
offset_snapshots <number>	Frequency of snapshot extraction. Every <number>th structure will be extracted from the trajectory.
trajectory_files all | <file>	Trajectory that shall be considered in snapshot extraction. For a consistent numbering and addressing of the snapshots request consideration of all trajectories by specifying `all`. The interval from which snapshots shall be extracted can be defined via the flags `first_snapshot`, `last_snapshot`, and `offset_snapshots`. If individual trajectories shall be used, specify each trajectory file in a separate line starting with the flag `trajectory_files`. Default = `all`.
snap_extract_template <file>	Optional: Absolute path and name of input-file for mm_pbsa.pl that shall be used for coordinate extraction. If no file is specified, it is assumed that the default file `examples/input_info/extract_snaps.in` shall be used.
image_trajectories 1 | 0	If set to 1, snapshots of the specified trajectories will be imaged to the origin before coordinate extraction. It is strongly recommended to use this option for all MM-PBSA / MM-GBSA calculations. Attention: Imaging may require a large amount of additional disc space. Default = 1.
use_imaged_trajectories 1 | 0	If imaged trajectories were generated in a previous FEW run, then these will be re-used for snapshot extraction if `use_imaged_trajectories=1`. In case imaged trajectories already exist and `use_imaged_trajectories=0` the existing trajectories will be renamed and new imaged trajectories will be generated from which then snapshots are extracted. Default = 1.

image_mass_origin 1 | 0	Optional: If set to 1, the receptor is imaged relative to the mass origin instead of the coordinate origin. Switching this flag on ensures compartibility of the imaging procedure with the one of the Amber FEW version. Default = 0.

MM-PBSA / MM-GBSA Analysis

PB 0 | 1 | 2 | 3 | 4	Type of Poisson-Boltzmann calculation (cf. Table 32.4 for an overview of the available calculation options). Please consider that only PB and GB methods requiring the same radii can be run together, i.e. `PB=4` and `GB=1`. All other PB methods can only be run with `GB=0`.
GB 0 | 1 | 2 | 5 | 6	Type of generalized Born calculation (cf. Table 32.4 for an overview of the available calculation options).
decomposition 0 | 1 | 2 | 3 | 4	If larger than 0 energy decomposition of the specified type is performed (cf. idecomp in Chapter 17 for decomposition options). Decomposition only works with `PB=4` and `GB=1`.
no_of_rec_residues <number>	Actual number of residues in the receptor structure.
total_no_of_intervals <number>	Total number of intervals that shall be analyzed. Please note that specifying more than one interval is only reasonable, if different offsets between structures shall be considered. Otherwise the energies for subsets of the analyzed snapshots can be calculated using the mm_pbsa_statistics.pl script provided in AMBER. Default = 1.
first_PB_snapshot <number>	Number of the first structure to be considered in the analysis.
last_PB_snapshot <number>	Number of the last structure to be considered in the analysis.
offset_PB_snapshots <number>	Offset between structures that shall be considered in the MM-PBSA / MM-GBSA analysis. Every <number>th snapshot will be taken into account.
mmpbsa_batch_template <file>	Absolute path and name of batch template file for the MM-PBSA / MM-GBSA calculations. Example file: `examples/input_info/MMPBSA.pbs`. Please adjust the template according to your computing environment, but keep everything from the section "Prepare calculation" onward and ensure that the line for job naming ends with "-N". The files generated during the calculation will be temporarily stored in the /tmp folder of the machine used for the calculation. Thus, not more than one node should be used per calculation.
mmpbsa_batch_path <path>	Optional: If the calculations shall be conducted using a different path than the one used for setup, this path can be specified here. In case no path is given the `<output_path>` will be used.
mmpbsa_sander_exe <file>	Optional: Absolute path and name of sander executable that shall be used instead of the default executable in $AMBERHOME/bin
parallel_mmpbsa_calc <number>	Number of processors to use in parallel run. This flag sets the PARALLEL flag in the mmpbsa.in file, i.e. <number> of threads will be run. Default = 1 (serial).
mmpbsa_template <file>	Optional: Absoulte path and name of the input file for mm_pbsa.pl that shall be used for the MM-PBSA / MM-GBSA calculations. If no file is specified, the default file located under `examples/input_info/mmpbsa.in` is taken. The default file can be modified by expert users, but only the following parameters may be changed: VERBOSE, DIELC, INDI, EXDI, SCALE, LINIT, ISTRNG, SALTCON, INTDIEL, and/or EXTDIEL.

Parameters for MM-PBSA calculations with implicit membrane

Implicit membrane MM-PBSA calculations are currently only possible if the Adaptive Poisson-Boltzmann Solver APBS [607–611] is installed on the system where the calculations shall be performed. Furthermore exclusively the combination `PB=3`, i.e. Poisson-Boltzmann calculation with Parse radii [157], and `GB=0`, i.e. no generalized Born calculation, is available (see options for `PB` and `GB` above). In addition, in order to avoid path inconsistencies, the setup of the calculations should be conducted with FEW on the same system were the calculations shall be run. The MM-PBSA calculations with implicit membrane are carried out with the Perl script `$AMBERHOME/AmberTools/src/FEW/micellaneous/mmpbsa_FEWmem.pl`. For the calculations also the files `apbs_mem_dummy.in` and `apbs_mem_solv.in` or `apbs_mem_dummy_focus.in` and `apbs_mem_solv_focus.in` provided in the `micellaneous` directory are required. Therefore the path of the FEW version used for the setup of the calculations should not differ from the path under which FEW can be found during the calculations. The parameters for the implicit membrane can be selected and tested with APBSmem (http://apbsmem.sourceforge.net) [612]. If you use the implicit solvent, implicit membrane MM-PBSA calculation functionality of FEW please cite APBS [607] as well as draw_membrane2 [613] and the extension of FEW for handling membrane systems [614].

membrane_residue_no <number>	Number of residues in the explicit membrane present in the MD simulation that serves as basis for the MM-PBSA calculation. Please consider all residues that are part of the membrane and not only the number of lipids. In the Lipid14 force field for example the lipids are split into head and tail groups, which are treated as separate residues.
implicit_membrane 1 | 0	If set to 1, an implicit membrane is considered in the MM-PBSA calculation, i.e. the system is embedded in an membrane slab with a lower dielectric constant than water.
apbs_executable 1 | 0	Full path to APBS executable, e.g. `/home/Software/iAPBS/bin/apbs`.
epsilon_solute 1 | 0	Dielectric constant of the solute, i.e. the protein and the ligand, in the MM-PBSA calculation. Please note, that the variable `DIELC` in the template input script for `mm_pbsa.pl` specified under `mmpbsa_template` needs to be set to the same dielectric constant to ensure that the calculated molecular mechanics electrostatic energies are scaled by the same constant.
bottom_membrane_boundary <no.>	Lower boundary of the membrane slab relative to the coordinate origin in [Å]. If more than one slab region is defined please give the lower boundary of the slab that is farthest away from the origin, see Figure 32.5. Default = -18 Å.
membrane_thickness <no.>	Thickness of the implicit membrane slab in [Å]. If a membrane slab with different slab regions is defined, please specify the thickness of the complete slab including all sub-slabs, see Figure 32.5. Default = 36 Å.
membrane_dielc <no.>	Dielectric constant of the implicit membrane slab. If a multi-slab membrane is constructed, this is the dielectric constant of the central membrane slab closest to the coordinate origin, see Figure 32.5. Default = 2.
second_slab_thickness <no.>	Optional: Thickness of a second slab region flanking the central slab on both sides. This slab can e.g. be used to model the properties in the region of or close to the lipid head groups. Please not that the thickness of the central slab defined under `membrane_thickness` decreases by 2 × `second_slab_thickness`, see Figure 32.5.

second_slab_dielc <no.>	Optional: Dielectric constant of the two second implicit membrane slab regions above and below the central membrane slab (Figure 32.5). This dielectric constant is usually larger than `membrane_dielc` to describe the properties in the region of or close to the lipid head groups. For a discussion of the complex electrostatic properties of a lipid bilayer see e.g. [615][616].
third_slab_thickness <no.>	Optional: Thickness of a third slab region located between the central slab and the second slab on both sides of the central slab (Figure 32.5). Please not that the thickness of the central slab defined under `membrane_thickness` decreases by (2 × `second_slab_thickness`) + (2 × `third_slab_thickness`), see Figure 32.5.
third_slab_dielc <no.>	Optional: Dielectric constant of the third implicit membrane slab region sandwitched between the second slab region and the central slab on both sides of the central slab (Figure 32.5). This dielectric constant is usually larger than `membrane_dielc` to describe the properties of the membrane region close to the membrane surface. For a discussion of the complex electrostatic properties of a lipid bilayer see e.g. [615][616].
ion_concentration <no.>	Concentration of ions, i.e. salt, that shall be considered in the Poisson-Boltzmann calculation. Default = 0.15 M.
upper_exclusion_radius <no.>	Upper exclusion radius in [Å]. See [612] and Figure 32.5.
lower_exclusion_radius <no.>	Lower exclusion radius in [Å]. See [612] and Figure 32.5.
do_focussing 1 | 0	Perform a three step APBS focussing calculation. In such a calculation three successive calculations are performed starting from a large grid followed by focussing using smaller grids, see http://www.poissonboltzmann.org. Default: 0.
size_large_grid <no.>	Optional: Size of the largest grid in the focussing calculation in [Å]. This is only considered if `do_focussing=1`. Please choose the size of the grids such that even the smallest grid (`size_small_grid`) completely comprises the membrane slab in the direction orthogonal to the plane of the membrane slab. Default = 300 Å.
size_medium_grid <no.>	Optional: Size of medium grid in the focussing calculation in [Å]. This is only considered if `do_focussing=1`. Please choose the size of the grid such that even the smallest grid (`size_small_grid`) completely comprises the membrane slab in the direction orthogonal to the plane of the membrane slab. Default = 200 Å.
size_small_grid <no.>	Size of the grid, if `do_focussing=0`, or size of the smallest grid, if `do_focussing=1`, in [Å]. Please choose the size of the grids such that it completely comprises the membrane slab in the direction orthogonal to the plane of the membrane slab. Default = 100 Å..
grid_dimensions <no.>	Number of grid points in each dimension, i.e. x, y, and z directions, of the grid. Valid values are 97, 129, and 161. Defaults: If `do_focussing=0` then `grid_dimensions=161` and if `do_focussing=1` then `grid_dimensions=97`.

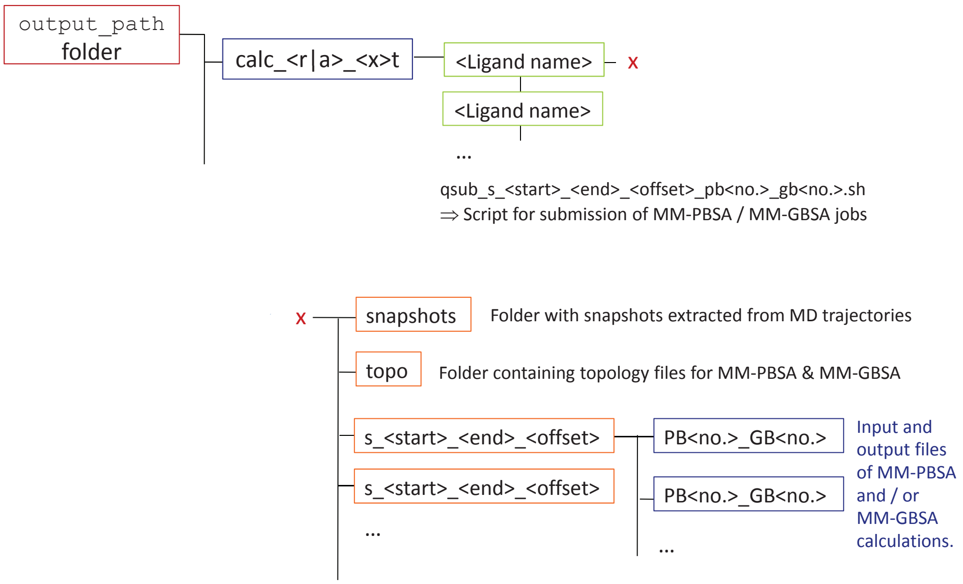

output_path
folder
calc_<r|a>_<x>t
<Ligand name>
x
<Ligand name>
...
qsub_s_<start>_<end>_<offset>_pb<no.>_gb<no.>.sh
⇒ Script for submission of MM-PBSA / MM-GBSA jobs
x
snapshots
Folder with snapshots extracted from MD trajectories
topo
Folder containing topology files for MM-PBSA & MM-GBSA
s_<start>_<end>_<offset>
s_<start>_<end>_<offset>
...
PB<no.>_GB<no.>
PB<no.>_GB<no.>
...
Input and output files of MM-PBSA and / or MM-GBSA calculations.

Figure 32.4.: *Folder structure and files created during setup of MM-PB/GBSA calculations.*

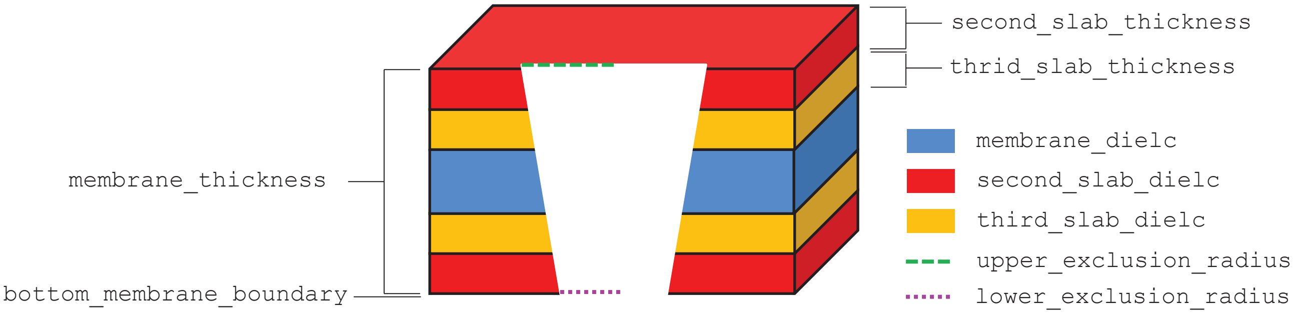

second_slab_thickness
thrid_slab_thickness
membrane_thickness
bottom_membrane_boundary
membrane_dielc
second_slab_dielc
third_slab_dielc
upper_exclusion_radius
lower_exclusion_radius

Figure 32.5.: *Parameters for definition of implicit membrane in MM-PBSA calculations.*

Postprocessing:

If MM-PBSA or MM-GBSA calculations without decomposition were conducted for several ligands, the binding free energies and important energetic contributions can be extracted from the `<ligand>_statistics.out` files created by mm_pbsa.pl using the script ...`/FEW/micellaneous/extract_WAMMenergies.pl`.

Usage:

```
perl   extract_WAMMenergies.pl   <structure file>   <path>   pb<no.>_gb<no.>   <Start>_<Stop>_<Offset>
```

structure file	Text file containing names of ligands for which energies shall be extracted; one name per line.

path	Path to directory containing MM-PBSA or MM-GBSA results, e.g., `/home/<user>/work_dir/calc_r_1t`.
pb<no.>_gb<no.>	FEW internal number of type of MM-PBSA / MM-GBSA calculation; see Table 32.4. The script can be used for all types of implicit solvent calculations available in FEW, except the hybrid model (PB2) and decomposition (dec).
<Start>_<Stop>_<Offset>	Snapshots taken into account in the MM-PBSA / MM-GBSA calculations; see flags `first_PB_snapshot`, `last_PB_snapshot`, and `offset_PB_snapshots` in the "MM-PBSA / MM-GBSA Analysis" section above.

A file called pb<no.>_gb<no.>.txt will be created in the current working directory. In this file the electrostic (ELE), van der Waals (VDW), nonpolar solvation (NP_SOLV), and polar solvation (P_SOLV) energy contributions to binding as well as the total binding free energy (ETOT) are listed for each ligand.

32.5. Linear interaction energy workflow (LIEW)

The LIE workflow enables energy calculations according to the linear interaction energy approach introduced by Åquist et al. [617] and was applied in numerous ligand binding affinity studies [618–620]. In this approach the changes upon complex formation in the electrostatic and the van der Waals interaction energy between a ligand and its surrounding environment are calculated based on MD simulations of the receptor bound ligand and of the ligand in solution. The binding free energy is estimated by combining differences in the electrostatic and van der Waals interaction energies in a linear equation with the coefficients α and β and possibly a constant term γ.

$$\triangle E^{LIE} = \beta\left(E^{ele}_{bound} - E^{ele}_{free}\right) + \alpha\left(E^{vdW}_{bound} - E^{vdW}_{free}\right) + \gamma$$

Commonly β is set to 0.5. However, several alternative strategies for selecting the coefficients and γ exist [618, 621, 622]. Furthermore it has been proposed to consider the difference in solvent accessible surface area between the bound and the free state of the ligand in the calculation of the binding free energy [623, 624].

$$\triangle E^{LIE} = \beta\left(E^{ele}_{bound} - E^{ele}_{free}\right) + \alpha\left(E^{vdW}_{bound} - E^{vdW}_{free}\right) + \gamma(SASA_{bound} - SASA_{free})$$

With the LIE workflow it is possible to setup the required MD simulations and to calculate the electrostatic and van der Waals interaction energy contributions as well as the solvent accessible surface area based on snapshots from the MD simulations by an automated procedure. This enables a fast calculation of the energy components needed for a LIE analysis, making energetic calculations for multiple ligands feasible. The computed energies can be used to construct a LIE model employing a (multiple) linear regression analysis.

The MD simulations can be conducted with *sander* or *PMEMD* of Amber. Electrostatic and van der Waals interaction energies of the ligand based on snapshots from the MD simulations are calculated with *sander*.

MD simulations for LIE analysis can be prepared using the common MD setup functionality of FEW described in section 32.3. Only MD setups according to the 3-trajectory approach are possible when the LIE procedure is requested. The receptor part of the 3-trajectory approach will automatically be neglected such that only files for the two simulations required for LIE analysis are generated. Thus, internally a 2-trajectory approach is prepared.

The availability of output/trajectory-files of simulations of the receptor bound ligand and of the ligand free in solution in the folders created by the MD setup procedure of FEW is a prerequisite for the energetic calculations. As for all FEW setup procedures, the command file for the energetic calculations according to the LIE approach needs to contain the flags specifying the input and output directories and formats (see section 32.3 "Common setup of molecular dynamics simulations" for a detailed explanation) as well as procedure specific flags. Example command files for LIE calculation setup are provided in `$AMBERHOME/AmberTools/src/FEW/examples/command_files/LIE`. An overview of the folder structure created by the LIE workflow is shown in Figure 32.6.

Specification of input / output directories and formats

lig_struct_path <path>	Path to folder containing the ligand structures. All ligand structures should be available as single structure mol2 files, because the format conversion should have been carried out in the preparatory step.
output_path <path>	Path to the basis output directory. This path needs to be identical to the `<output_path>` specified in the common MD setup step.
water_in_rec 0 | 1	Set to 1 if crystal water molecules were present in the receptor structure used for MD simulation setup.

General parameters for LIE calculations

lie_calc 0 | 1	Request setup of LIE calculations.
charge_method am1 | resp	Charge method that shall be considered for LIE analyses. Trajectories of MD simulations with corresponding atomic charges for the ligand need to be available. The generation of the files required for these simulations is described in section 32.3.
no_of_rec_residues <number>	Actual number of residues in the receptor structure.
additional_library <file>	Optional: Absolute path and name of additional library file. Such a library file is only required if the receptor structure contains non-standard residues.
additional_frcmod <file>	Optional: Absolute path and name of additional parameter file. Such a parameter file is only required if not all parameters that are needed to describe the receptor are available in the ff12SB force field.
lie_executable <executable>	Optional: Absolute path and name of the LIE.pl program for calculation of interaction energies according to the LIE approach, which is distributed with FEW. If no executable is specified, it is assumed that the LIE program can be found under the default path and name at `$AMBERHOME/AmberTools/src/FEW/miscellaneous/LIE.pl`
lie_batch_template <file>	Absolute path and name of batch file for LIE analysis. An example file can be found under `exmaples/input_info/lie.pbs`. Please adapt the batch file according to the requirements of your queuing system, but do not change anything from the "Prepare calculation" section onward and ensure that the line for job naming ends with "-N".
lie_batch_path <path>	Optional: Path that shall be used instead of the `<output_path>` for the setup of batch file. This information is only required if the LIE analysis shall be run under a different path than the setup.

Snapshot extraction

snaps_per_trajectory <number>	Number of snapshots per trajectory. If more than one trajectory file is provided, all trajectory files need to contain the same number of snapshots.
image_trajectories 1 | 0	If set to 1, the structures will be imaged to the origin before coordinates are extracted. This is strongly recommended. However, please regard that imaging may consume a large amount of disc space, since new trajectories with imaged structures are created. Default=1.
trajectory_files all | <file>	Trajectory files that shall be regarded. For a consistent numbering of the snapshots it is strongly recommended to consider all trajectories that have been generated by specifying `all`. Subsets of snapshots that shall be considered in the energy calculation can be selected by the parameters `first_lie_snapshot`,

`last_lie_snapshot`, and `offset_lie_snapshots`. Individual trajectory files can be selected by specifying their file name (without the path). Each file that shall be considered must be specified in a separate line starting with the keyword `trajectory_files`. Default = `all`.

LIE Analysis

first_lie_snapshot <number>	No. of first snapshot that shall be regarded in the energy calculation.
last_lie_snapshot <number>	No. of last snapshot that shall be regarded in the energy calculation.
offset_lie_snapshots <number>	Offset between snapshots that shall be regarded in the energy calculation. Every <number>th snapshot will be considered.
calc_sasa 0 | 1	Request calculation of solvent accessible surface area. Default = 0.
sander_executable <executable>	Optional: Absolute path and name of *sander* executable that shall be used for the energy calculation if not the default application under $AMBERHOME/bin shall be employed.
parallel_lie_call <call>	The calculations can be conducted using a parallel version of *sander*. If you would like to start a parallel job, please specify the call required for starting a parallel execution of *sander* on your system here, e.g.: `mpirun -np 2`. Prerequisite for parallel execution: Parallel version of *sander* available.
delete_lie_trajectories 0 | 1	As storing the coordinates of the structures in a form specifically required for LIE analyzes can consume a large amount of disc space, it can be advantageous to only temporarily create them. If `delete_lie_trajectories` is set to 1, the trajectories for LIE analyzes are deleted directly after the energy calculations.

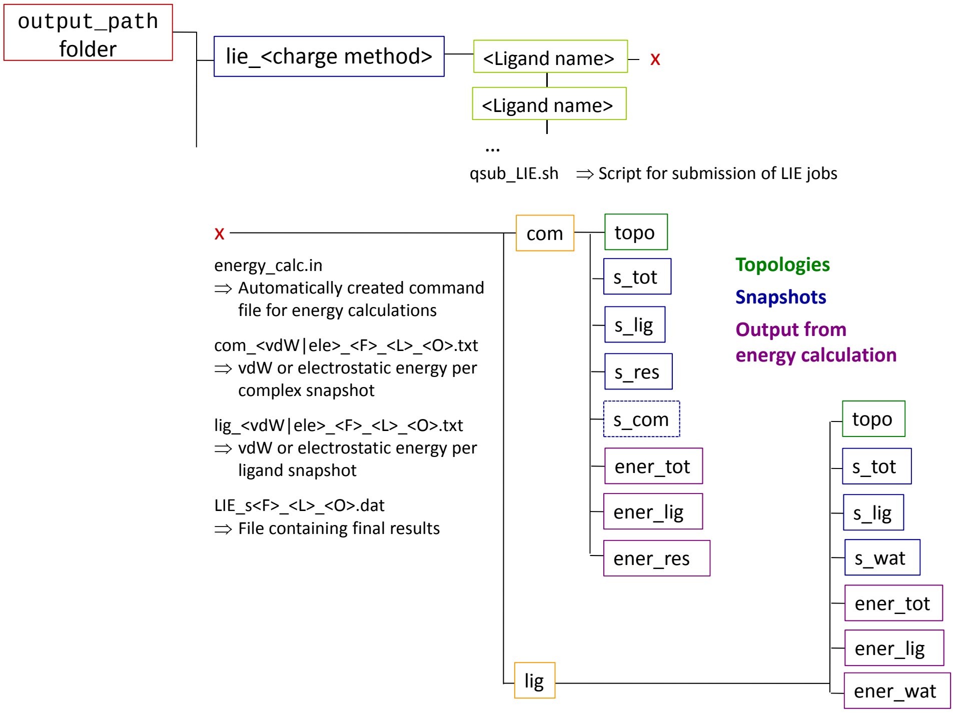

output_path folder
lie_<charge method>
<Ligand name>
x
<Ligand name>
...
qsub_LIE.sh ⇒ Script for submission of LIE jobs
x
energy_calc.in
⇒ Automatically created command file for energy calculations
com_<vdW|ele>_<F>_<L>_<O>.txt
⇒ vdW or electrostatic energy per complex snapshot
lig_<vdW|ele>_<F>_<L>_<O>.txt
⇒ vdW or electrostatic energy per ligand snapshot
LIE_s<F>_<L>_<O>.dat
⇒ File containing final results
com
topo
s_tot
s_lig
s_res
s_com
ener_tot
ener_lig
ener_res
Topologies
Snapshots
Output from energy calculation
topo
s_tot
s_lig
s_wat
ener_tot
ener_lig
ener_wat
lig

Figure 32.6.: *Folder structure and files created during the setup of LIE calculations. For reasons of clarity* `first_lie_snapshot`, `last_lie_snapshot`, *and* `offset_lie_snapshots` *were replaced by aliases F, L, and O.*

Postprocessing:

If LIE analyzes were conducted for several ligands, the differences in electrostatic and vdW interaction energies can be extracted from the `LIE_s<first>_<last>_<offset>.txt` files using the script `$AMBERHOME/AmberTools/src/FEW/miscellaneous/extract_LIEenergies.pl`.

Usage:

```
perl extract_LIEenergies.pl <structure file> <path> <name of LIE output file>
```

structure file Text file containing the names of the ligands that shall be considered (one ligand per line) and experimentally measured IC50 or K_i or binding free energies in tab-separated format.

Example:

#Ligands	dG
Lig_5	-0.5394
Lig_17	-1.3409

path Path to directory containing the LIE results, e.g. `/home/<user>/work_dir/lie_am1`.

name of LIE output file	Name of the final result file of the LIE calculations, i.e. `LIE_<first>_<last>_<offset>.txt`, where `<first>`, `<last>`, and `<offset>` are equivalent to the values selected for the corresponding `<X>_lie_snapshot(s)` keywords described above.

A file called LIE_results.txt will be created in the current working directory. In this file, besides the ligand name and the binding affinity value provided in the `<structure file>`, the differences in electrostatic (ELE) and van der Waals (VDW) interaction energies and the difference in solvent accessible surface area between the bound and the free state are listed. The file can be used directly to derive a linear model by a (multiple) linear regression analysis.

32.6. Thermodynamic integration workflow (TIW)

The TI workflow enables a fast setup of transformation simulations between two ligands for the determination of the difference in free energy of binding according to the thermodynamic integration approach. Transformation simulations are prepared employing the one step, soft core option provided in AMBER. For a detailed description of the method see Section 21.1. Prerequisite for conducting the TI calculation setup with FEW is a parallel installation of the program *sander* of AMBER.

Transformation simulations can either be started from provided structures or from structures that have been pre-equilibrated with FEW. Equilibrated structures of complex and ligand can be prepared using the common MD setup functionality of FEW. See section 32.3 for details on how to prepare the files for minimization and equilibration. Alternatively, the TI setup can be requested based on coordinate and topology files generated from a crystal structure or from other sources. This option can be valuable in cases where high resolution crystal structures are avaible for the receptor bound state of both the initial (*V0*) and the final (*V1*) ligand and these show only marginal differences with respect to the receptor structure. In case user provided structures shall be used directly it is necessary to run the common MD setup procedure without providing the flag `MDequil_template_folder`, in order to prepare the files required for the TI calculations. Figure 32.7 illustrates the two setup options and the corresponding workflows.

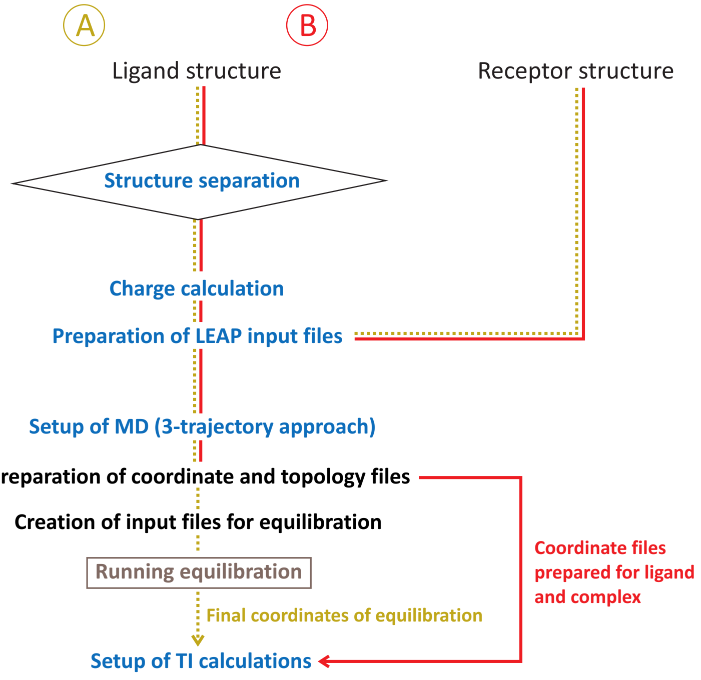

A
B
Ligand structure
Receptor structure
Structure separation
Charge calculation
Preparation of LEAP input files
Setup of MD (3-trajectory approach)
Preparation of coordinate and topology files
Creation of input files for equilibration
Running equilibration
Final coordinates of equilibration
Coordinate files prepared for ligand and complex
Setup of TI calculations

Figure 32.7.: *TI workflow options: TI based on (A) a structure equilibrated using the MD setup functionality of FEW or (B) a user provided structure, e.g., a crystal structure.*

The TI simulations are separated into a TI equilibration and a TI production phase. Input files for the latter can only be prepared when the equilibration simulations have been completed. In the equilibration phase the transformation simulations are conducted sequentially for all λ values in ascending order (Figure 32.8), i.e., the final coordinate file of the equilibration at the smallest λ value serves as input file for the next larger λ value, and so on. Thus, only one batch-job for the equilibration needs to be submitted per system.

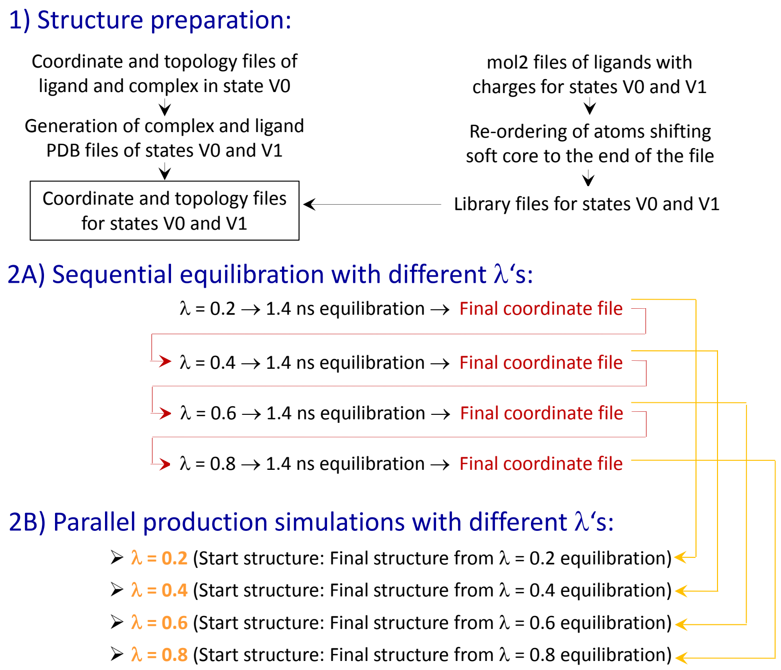

1) Structure preparation:
Coordinate and topology files of ligand and complex in state V0
Generation of complex and ligand PDB files of states V0 and V1
Coordinate and topology files for states V0 and V1
mol2 files of ligands with charges for states V0 and V1
Re-ordering of atoms shifting soft core to the end of the file
Library files for states V0 and V1
2A) Sequential equilibration with different λ's:
λ = 0.2 → 1.4 ns equilibration → Final coordinate file
λ = 0.4 → 1.4 ns equilibration → Final coordinate file
λ = 0.6 → 1.4 ns equilibration → Final coordinate file
λ = 0.8 → 1.4 ns equilibration → Final coordinate file
2B) Parallel production simulations with different λ's:
λ = 0.2 (Start structure: Final structure from λ = 0.2 equilibration)
λ = 0.4 (Start structure: Final structure from λ = 0.4 equilibration)
λ = 0.6 (Start structure: Final structure from λ = 0.6 equilibration)
λ = 0.8 (Start structure: Final structure from λ = 0.8 equilibration)

Figure 32.8.: *Internal TI workflow of FEW consisting of structure preparation, equilibration simulations, and production simulations.*

Production simulations are started from equilibrated structures, i.e., from coordinate files obtained in the equilibration phase. Prior to the setup of the production simulations it is checked whether the systems are thoroughly equilibrated employing a reverse cummulative averaging procedure [625]. The production simulations, which are prepared when the equilibration check is complete, can be conducted in parallel for all λ values. For each λ value a separate batch script is generated. Production simulations are run either until a convergence measure, calculated after each production step, falls below a specified limit or the total runtime defined in the command file is reached. Two alternative convergence criteria are available: (I) The difference between the current standard error in dV/dλ, determined according to [419], and the one calculated in the previous step. (II) The precision of dV/dλ, i.e., the expected deviation of the true mean from the sample mean determined based on a student's distribution at a confidence level of 95%. How often the convergence is checked depends on the simulation time specified in the provided template file for TI production. A convergence analysis is performed after each production run, and if the termination criterion is not reached, the next round of TI production is started. Since the calculation of the convergence measures requires the determination of the autocorrelation time in dV/dλ, the number of dV/dλ values that are written to the *sander* output file should be 10 times larger than the autocorrelation time. As the autocorrelation time is typically in the range of 1 ps [419], it is recommended to request writing of at least 20 dV/dλ values in the template production file when a recording interval of 1 ps is used. If the number of dV/dλ values is not larger than 10 times the autocorrelation time, the simulation procedure is terminated after the first production step. The convergence analysis is handled by the batch script and does not require user intervention.

When the transformation simulations have been completed (cf. Figure 32.9 for created folder structure), the TIW module of FEW can be used to calculate the difference in free energy of binding between the two studied ligands. The free energy difference $\triangle G$ is calculated by numerical integration over the average dV/dλ values obtained from the simulations at the individual λ's, employing the trapezoidal rule. The user can choose whether the commonly applied linear interpolation to λ=0 and λ=1 shall be conducted (eq. E1) or the boundary area of the dV/dλ curve shall be neglected (eq. E2).

$$\triangle G = \int_0^1 \langle dV(\lambda)/d\lambda \rangle \, d\lambda \quad \text{(E1)}$$

$$\triangle G = \sum \langle dV(\lambda)/d\lambda \rangle \, \triangle\lambda \quad \text{(E2)}$$

Finally the difference in free energy of binding $\triangle\triangle G$ is calculated by subtracting $\triangle G_{ligand}$ calculated based on the transformation of the ligand free in solution from $\triangle G_{complex}$ derived from the transformation within the complex (eq. E3).

$$\triangle\triangle G = \triangle G_{complex} - \triangle G_{ligand} \quad \text{(E3)}$$

The existence of files created according to the MD setup for the 3-trajectory approach with FEW is a prerequisite for the execution of the TI workflow. Example command files for TI calculation setup are provided in `$AMBERHOME/AmberTools/src/FEW/examples/command_files/TI`.

Specification of input / output directories and formats

lig_struct_path <path>	Path to folder containing the ligand structures. All ligand structures should now be available as single structure mol2 files because the conversion should have been carried out in the preparatory step.
output_path <path>	Path to the basis output directory. This path needs to be identical to the `<output_path>` specified in the common MD setup step.

TI simulations

Parameters that have to be specified and need to be identical in all subsequent TI setup runs for one system:

ti_simulation_setup 0 | 1	Request setup of files for TI simulation.
charge_method am1 | resp	Charge method that shall be used for the calculations. MD setup files or equilibrated structures generated with the corresponding charge method need to be available. See section 32.3 on how to generate these files.
lig_name_v0_struct <name>	Name of ligand in start state (V0). The name needs to be identical to the name used for the corresponding structure in the MD setup step, i.e. basename of mol2 file.
lig_name_v1_struct <name>	Name of ligand in end state (V1). The name needs to be identical to the name used for the corresponding structure in the MD setup step, i.e. basename of mol2 file.
lig_alias_v0 <alias>	Alias that shall be used for the ligand in the start state (V0). The alias serves, e.g., as ligand residue name and identifier for the TI simulation files and must consist of 3 characters.
lig_alias_v1 <alias>	Alias that shall be used for the ligand in the end state (V1). The alias serves, e.g., as ligand residue name and identifier for the TI simulation files and must consist of 3 characters.

softcore_mask_v0 <mask>	Soft core mask for state V0 to be used for AMBER "scmask" definition. Format: <V0_alias>@<atom>,<atom>,... For details about the format see Section 21.1.
softcore_mask_v1 <mask>	Soft core mask for V1 to be used for AMBER "scmask" definition. Format: <V1_alias>@<atom>,<atom>,... For details about the format see Section 21.1.

The following three steps are done by three consecutive calls of FEW according to Figure 32.3.

1. Creation of coordinate and topology files

prepare_match_list 0 | 1	Request setup of match list with atom correspondence information for none soft-core part of states V0 and V1. The list contains the atom names of corresponding atoms in the two states, in tab-separated format. In case the automatic matching fails, the list can also be created manually.
prepare_inpcrd_prmtop 0 | 1	Request setup of coordinate and topology files. The steps needed for preparation of coordinate and topology files for start and end states are only carried out if this flag is set to 1.
lig_inpcrd_v0 <file>	Coordinate file of solvated ligand start structure in *coordinate* or *restart (inpcrd, restrt)* format. Either the end structure of an equilibration simulation or a crystal / model structure can be provided. Please regard that in the later case the structure will be directly subjected to an equilibration MD, without previous minimization, heating and density adjustment. A significant longer equilibration run will be necessary in this case. *Attention:* The coordinate file must have been prepared with the common MD setup functionality of FEW.
com_inpcrd_v0 <file>	Coordinate file of the solvated complex start structure either in *coordinate* or *restart* format (cf. `lig_inpcrd_v0` flag).
lig_prmtop_v0 <file>	Topology file of the solvated ligand corresponding to the coordinate file specified under `lig_inpcrd_v0`.
com_prmtop_v0 <file>	Topology file of the solvated complex corresponding to the coordinate file specified under `com_inpcrd_v0`.
match_list_file <file>	Absolute path and name of a file containing atom correspondence information between states *V0* and *V1*. An example match-file can be found in `examples/input_info/match_list.txt`. This information must only be provided if the automated generation of the atom correspondence list (`prepare_match_list=1`) was not successful and the list was created manually.
SSbond_file <file>	Absolute path and name of file containing disulfide bridge definitions for the receptor. For an example file see `examples/input_info/SSbridges.txt`
chain_termini <no.>,<no.>,....	Numbers of terminal residues of chains in receptor structure, e.g., if a chain ends at residue 234 and a new chain starts with residue 235, the number 234 needs to be specified as <no.>.
create_sybyl_mol2 0 | 1	Optional: Request generation of mol2 ligand files for V0 and V1 with sybyl atom types. As most molecule visualization programs support this format, the created files allow an easy comparison of atom names of start and end structures to check the correctness of the atom matching step.
additional_library <file>	Optional: Absolute path and name of additional library file containing information about non-standard residues or ions.

additional_frcmod <file>	Optional: Absolute path and name of additional parameter file. Such a file is only required if parameters necessary for the description of the receptor are missing in the ff12SB force field.

2. General parameters for preparation of TI transformation simulations

ti_batch_path <path>	Optional: If the simulations shall be run under a different path than the setup, a new `<output_path>` for the batch file generation can be specified.
ti_prod_template <file>	Optional: Template file for TI production simulations. Per default the example file under `examples/input_info/MD_prod_TI.in` will be used as a template. Please adapt the file according to your needs but keep the format of the lines containing the flags "t", "scmask", and "clambda". If decomposition is requested, please also use the format shown in the example file for the specification of "RES" and "LRES".
no_shake 0 | 1	Optional: Request calculation without AMBER shake option. In this case ensure that shake is also switched off in the `ti_prod_template` file. For an example file see `examples/input_info/MD_prod_noShake_TI.in`. It is recommended to conduct transformations not involving exchanges of atoms in rings or exchanges of single hydrogen atoms with shake (`no_shake=0`) on hydrogens (`ntc=2`, `ntf=2`) to be able to increase the integration step size to 2 fs. Default = 0.

A) Setup of scripts for TI equilibration

ti_equil 0 | 1	Request setup of files for TI equilibration.
ti_equil_batch_template <file>	Template batch file for the submission of the equilibration phase job to a queuing system. An example file can be found under `examples/input_info/equi_TI.pbs`. Please adapt the file according to the needs of your queuing system, but keep everything from the section entitled "Fix variables" up to the section "Re-queue" and ensure that the line for job naming ends with "-N".
ti_equil_lambda <no.>,<no.>,...	λ values for which TI equilibration calculations shall be prepared in ascending order. Please specify only the decimal digits, e.g. 1 for lambda 0.1, 05 for lambda 0.05. λ values can be in the range 0.01 – 0.99, i.e., 01 – 99 in the FEW internal nomenclature. For equilibration only equi-distant λ values can be used, i.e., $\triangle\lambda$ needs to be equal for all successive λ's.
ti_equil_template <file>	Template file for equilibration part of the equilibration phase. For an example file see `examples/input_info/equi_TI.in`. The equilibration part is followed by a 1 ns free MD simulation for finishing equilibration of the system. For setup of this later part the template file specified under `ti_prod_template` will be used.

B) Setup scripts for TI production simulations

ti_production 0 | 1	Request setup of scripts for TI production simulations. Please note that this option requires the presence of the results of the TI equilibration calculations in the corresponding "equi" folder.
ti_prod_lambda <no.>,<no.>,...	λ values for which TI production calculations shall be prepared in ascending order. Please specify only the decimal digits, e.g., 1 for lambda 0.1, 05 for lambda 0.05. λ values can be in the range 0.01 - 0.99, i.e. 01 - 99 in the FEW internal nomenclature.

total_ti_prod_time <time>	Total simulation time per λ value in [ns]. The number of cyclic runs required will be calculated based on the definitions in the `ti_prod_template`. Please ensure that the MD total simulation time is a multitude of the MD simulation time specified in the production template file. The requested total simulation time will only be reached, if the error limit for simulation termination is not reached before.
ti_prod_batch_template <file>	Template batch file for the submission of the production phase job to a queuing system. An example file can be found under `examples/input_info/prod_TI.pbs`. Please adapt the file according to your queuing system, but keep everything from the section entitled "Fix variables" up to the section "Re-queue" and ensure that the line for job naming ends with "-N".
converge_check_script <file>	Optional: Absolute path and name of Perl-script used for convergence checking after each production step. If the location of the script is not provided it will be assumed that the script is located under the default location at `.../FEW/miscellaneous/convergenceCheck.pl`
converge_check_method 1|2	Optional: Method that shall be used for convergence analysis. 1: Difference in standard error of dV/dλ between consecutive production runs; 2: Precision of dV/dλ determined employing student's distribution. For a detailed explanation refer to the introduction section of the TI calculation module (Section 32.6). Default = 1.
converge_error_limit <limit>	Optional: Error limit that shall be used as termination criterion for the TI production simulations. Default: 0.01 kcal/mol (method 1); 0.2 kcal/mol (method 2). As long as the convergence measure is larger than this limit and the total simulation time has not been reached, the simulation will go on.

3. Calculation of the difference in free energy of binding

$\Delta\Delta G_{\text{binding}}$ can be calculated using a command file containing the following parameters (in addition to the section specifying input / output directories and formats).

ti_ddG 0 | 1	Request calculation of the difference in free energy of binding between start (V0) and end (V1) ligands.
charge_method am1 | resp	Charge method (see above).
lig_name_v0_struct <name>	Name of ligand in the start state (V0). The name needs to be identical to the name used in the setup of the simulations (see above).
lig_name_v1_struct <name>	Name of ligand in the end state (V1). The name needs to be identical to the name used in the setup of the simulations (see above).
lig_alias_v0 <alias>	Alias that shall be used for the ligand in the start state (V0). The alias must be identical with the alias used for the setup of the TI simulations (see above).
lig_alias_v1 <alias>	Alias that shall be used for the ligand in the end state (V1). The alias must be identical with the alias used for the setup of the TI simulations (see above).
dVdL_calc_source <no.>-<no.>	Range of files from the production phase of the TI simulations that shall be considered in the calculation of the difference in free energy of binding. If set to "0", all recorded files will be considered. If only files in a certain range shall be regarded, specify the range, e.g., the range "3-5" will

	result in considering of the files xxx_prod03_v1.out, xxx_prod04_v1.out and xxx_prod05_v1.out from the production run of the TI simulations. In case all files from a certain time point onward shall be regarded, provide a range that ends with zero, e.g. "4-0".
ddG_calc_method 0 | 1	Method that shall be used for the calculation of $\Delta\Delta G_{binding}$. If “1” is specified, the common procedure with linear interpolation to λ=0 and λ=1 is used. In case of "2", no linear interpolation is conducted. The later calculation method can only be used, if the production simulations were run with equi-distant λ values, i.e., $\triangle\lambda$ was of the same size for all successive λ’s.

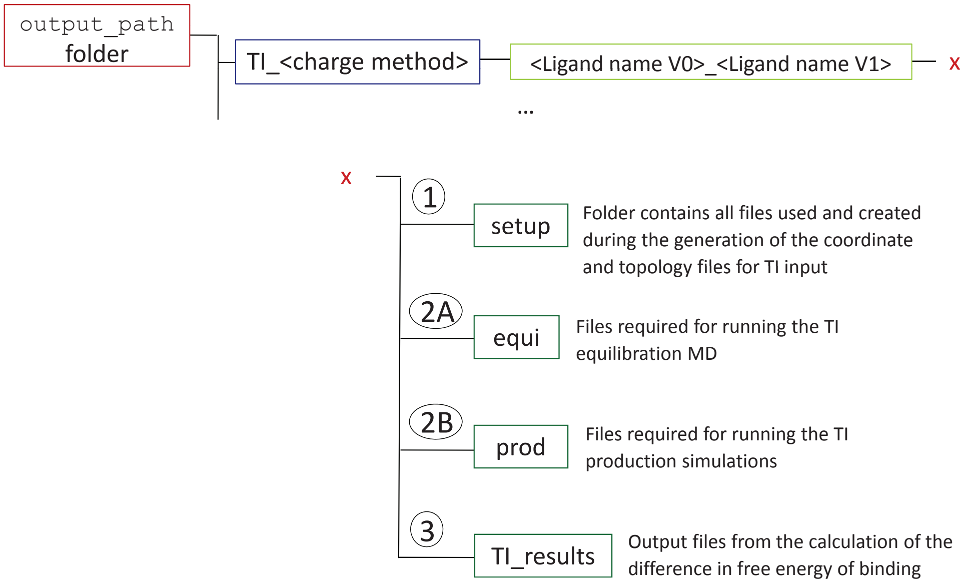

output_path
folder
TI_<charge method>
<Ligand name V0>_<Ligand name V1>
x
...
x
1
setup
Folder contains all files used and created during the generation of the coordinate and topology files for TI input
2A
equi
Files required for running the TI equilibration MD
2B
prod
Files required for running the TI production simulations
3
TI_results
Output files from the calculation of the difference in free energy of binding

Figure 32.9.: *Folder structure and files generated during TI calculation setup with FEW. Numbering according to steps shown in Figure 32.8.*

Miscellaneous:

Script for the identification of “optimal” transformations

If several ligands shall be studied by thermodynamic integration the shortest path algorithm of Kruskal [626] can be used to determine the ”optimal” transformations between the ligands, i.e. those that require overall the smallest structural changes. In this way, relative binding free energies can be computed between those ligand pairs that show overall the highest similarity. A script called `identify_transformations.pl` provided in the miscellaneous folder of FEW can be used to identify the ”optimal” transformations. This script employs Kruskal’s algorithm to determine those transformations that lead to the smallest overall score based on a matrix of similarity scores. Such matrix of similarity scores can for example be obtained by a pairwise ligand comparison employing the Tanimoto-Combo score of ROCS [627, 628]. The script uses the Perl module Graph::Kruskal, which needs to be downloaded from CPAN (http://www.cpan.org/modules/index.html) and installed as part of the local Perl installation before the script `identify_transformations.pl` can be used.

Usage:

```
perl identify_transformations.pl <number of structures> <score matrix file>
```

number of structures

Integer number specifying the number of structures that shall be regarded in the search for "optimal" transformations.

score matrix file

Absolute path and name of file containing the score matrix based on which the "optimal" transformations are determined. This matrix file should be in tab-separated text format. The similarity matrix needs to comprise N × N score values, where N is the number of ligands that shall be regarded. It is assumed that smaller scores correspond to a higher similarity between ligands. The first line and the first column should contain the names of the ligands.

Example - section of score matrix file:

```
      L01   L02   L03   L04   ...
L01   0     217   284   199   ...
L02   217   0     285   427   ...
L03   284   285   0     118   ...
L04   199   427   118   0     ...
...   ...   ...   ...   ...
```

33. XtalAnalyze

XtalAnalyze is a collective name for a set of scripts that facilite analysis of crystal trajectories. The analysis of crystal trajectories presents unique challenges because the system is composed of multiple independent copies of the molecule related by crystal symmetry in an unrestrained crystal lattice. XtalAnalyze allows the user to obtain basic statistics such as RMSD and B-factors for the unique asymmetric unit using data from the entire supercell. Other components include a quick plotting utility, a tool to calculate average electron density from the simulation and a set of smaller utilities for various tasks commonly encountered when working with crystal simulations.

33.1. XtalAnalyze.sh

XtalAnalyze.sh performs basic analysis of crystal trajectories. Generally, it fits each frame of the trajectory to the original experimental crystal supercell by center of mass fitting, decomposes the trajectory into the individual asymmetric units and applies the corresponding space group's crystal symmetry and translation operations in reverse to bring the coordinates of each unit back into the space of the original asymmetric unit (asu) from which the supercell was first created. Calculation of various statistics such as lattice and asu RMSD, reverse symmetry and rmsd B-factors and average coordinates are then performed. In addition, the individual asymmetric unit trajectories are retained allowing for more specialized analysis by the user of select parts of the supercell trajectory. For a more detailed discussion of the methods employed and in particular of the meaning of terms such as 'lattice and asu RMSD', 'revsym B-factors', see Ref. [629]. To run XtalAnalyze.sh the user should have Python 2.7, Numpy (*http://www.numpy.org/*) and ScientificPython (*http://dirac.cnrs-orleans.fr/plone/software/scientificpython/*).

`XtalAnalyze.sh` is a wrapper for cpptraj and python scripts found in $AMBERHOME/AmberTools/src/xtalutil/Analysis. To run it, the user should copy `XtalAnalyze.sh` and `ReadAmberFiles.py` into a working directory of their choosing. They should then open `XtalAnalyze.sh` in a text editor and modify the variables found in the header under the "USER: SET VARIABLES" heading.

```
#======================================================================#
#                                                                      #
#  USER: SET VARIABLES                                                 #
#======================================================================#
#paths may be absolute or relative to working directory
SC_PRMTOP=topology_file.prmtop
SC_TRAJECTORY=trajectory_file.nc
SC_REFERENCE=restart_file.rst7
PDB_FILE=pdb_file.pdb
SC_PROP="3 2 2"
ASU_NRESIDUES=139
Timestep=0.1

# Amber masks for b-factor and rmsd calculations. These can be modified
# according to the user's needs.
BM1=":1-129@CA"
BM2=":1-129&!(@H=,CA,C,O,N)"
RM1=":1-129&!(@H=)"
RM2=":1-129@CA,C,N"
```

SC_PRMTOP - the topology file used for the crystal simulation

SC_TRAJECTORY - simulation trajectory file in NetCDF format; the user may first wish to remove waters, solvent ions, etc. to speed up calculations

SC_REFERENCE - original coordinates of the supercell built from the experimental asymmetric unit (usually using UnitCell and PropPDB) in Amber restart format. This is the supercell coordinates previous to any minimization or dynamics. All of the asymmetric units in this file should be identical copies of each other, and the coordinates of the original asymmetric unit must be exactly the same as in the experimental PDB file. The box-size (last line of the file) must correspond to the supercell not the crystal unit cell.

PDB_FILE - PDB file of the asymmetric unit which MUST contain SMTRY and CRYST1 records

SC_PROP - three integers in quotes corresponding to the x, y and z propagations used to build the supercell

ASU_NRESIDUES - number of residues in the asymmetric unit

Timestep - timestep between frames (in units of user's choosing). This is used for the RMSD output.

BM1 - individual atomic B-factors will be calculated for each atom in this mask. (See Section 19.1 for Amber mask format.) Usually this is done for the C-alpha atoms in a protein crystal but can be set to any selection of the user's choosing.

BM2 - average per residue B-factors will be calculated for all the atoms in this mask. Usually this is used for to calculate the average side-chain B-factor of each residue but can be set to any selection of the user's choosing.

RM1,RM2 - XtalAnalyze will calculate two sets of RMSDs according to the two masks provided. Usually this is done for the set of all heavy (non-hydrogen) atoms and for only the backbone atoms, but the masks can be modified to any selection of the user's choosing.

Once the correct values are set, save the file and run it from command line:

```
[]$ chmod +x XtalAnalyze.sh
[]$ ./XtalAnalyze.sh
```

Most important output produced by `XtalAnalyze.sh` is:

$WD- the working directory from which XtalAnalyze was run

$WD/asu.prmtop;$WD/asu.pdb;$WD/asu.rst7 topology, restart and pdb filesfiles corresponding to single asymmetric unit with experimental coordinates

$WD/volume.dat,$WD/volume.txt volume of supercell at each frame and percent of experimental volume; summary of basic volume statistics

$WD/RMSDUC.dat matrix of rmsd of each asu average structure to each other asu

*$WD/revsym/AvgCoord*rst7* 'asu' and 'lattice'-fit average structure

*$WD/revsym/AvgCoord*dat* RMSD of the average structures to experimental

*$WD/revsym/bfac*dat* 'revsym'(=lat) and 'rmsd'(=asu) B-factors for BM1(=calpha) and BM2(=sdch) selections

$WD/revsym/RevSym_%d_%d.nc individual asu trajectories after applying crystal symmetry and translation in reverse

*$WD/revsym/rmsd*table.dat* lattice and asu RMSD's for RM1(=heavy) and RM2(=backbone) atom selections. Contains the mean RMSD and standard deviation over all asymmetric units (first two columns) followed by one column for the RMSD of each individual asymmetric unit

$WD/splittrajectories/ this directory contains trajectories of each individual asymmetric unit before applying symmetry/translation operations in reverse as well as average structures and B-factors for each asymmetric unit individually as well as an RMSD matrix comparing all asymmetric units to one another

33.2. XtalPlot.sh

`XtalPlot.sh` creates plots of the data output by `XtalAnalyze.sh`. It is also used by copying into a directory of the user's choosing, modifying the variables under "USER: SET VARIABLES", and excuting via command line. `XtalPlot.sh` requires that the user have Python 2.7 with Matplotlib (*http://matplotlib.org*) and Numpy (*http://www.numpy.org/*). The variables to set are the following:

```
#=======================================================================#
#                                                                       #
#  USER: SET VARIABLES                                                  #
#=======================================================================#
PLOT_PFX="4lzt"                      # filename prefix (for plots, etc.)
BFAC_CRYST_CALPHA=calpha.bfactors   # experimental Bfacs
BFAC_CRYST_SDCH=residue.bfactors
SC_REFERENCE=4lztSc_centonpdb_nowat.rst7 #same as XtalAnalyze.sh
PDB_FILE=4lzt.pdb                        #same as XtalAnalyze.sh
SC_PROP="3 2 2"                          #same as XtalAnalyze.sh
ROWS=4                                   #number of panel rows to use on individual asu plots
COLS=3                                   #number of panel cols to use on individual asu plots
```

SC_REFERENCE,PDB_FILE,SC_PROP - same as for XtalAnalyze.sh (see above)

PLOT_PFX - in quotes, a string used in the titles of plots. Usually the PDB code or other identifier familiar to the user.

ROWS,COLS - the number of rows and columns to use for single-page,multi-panel plots of individual asymmetric unit statistics (for example, if the simulated supercell has 12 asymmetric units, a good choice would be ROWS=3 COLS=4).

BFAC_CRYST_CALPHA,BFAC_CRYST_SDCH - text files containing the experimental BM1 and BM2 B-factors. These will be plotted for comparison with the simulation calculated B-factors. The files should be in plain text format with the B-factors, one for each atom in the BM1 mask or each residue in the BM2 mask per line in the SECOND column. Contents of other columns are disregarded. Example:

```
1  11.32    CA     1
2   8.52    CA     2
3   7.86    CA     3
4   8.74    CA     4
5   9.46    CA     5
6   8.44    CA     6
...
```

These files can be created using the script `GetBfactors.py` in $AMBERHOME/AmberTools/src/xtalutil/Analysis/ which is run on the command line like this:

```
[]$ ./GetBfactors.py [original.pdb] [amber.pdb] [nres]
```

where *original.pdb* is the experimental pdb file of the asymmetric unit with B-factors, *amber.pdb* is the pdb file of the asymmetric unit with Amber atom order (usually *asu.pdb* produced by XtalAnalyze.sh) and *nres* is the number of residues in the asymmteric unit. Running `GetBfactors.py` will produce the text files *calpha.bfactors* and *sdch.bfactors* which can be used for the BFAC_CRYST_CALPHA and BFAC_CRYST_SDCH settings above. See the header of the script for a more detailed explanation. Note that due to the complexities of the PDB format the user should always inspect the output and make sure that the B-factors in the output file do in fact correspond to the atoms in *asu.pdb*.

`XtalPlot.sh` creates a new directory named *plots* and places all newly created plots, in pdf format, in it.

33.3. md2map.sh

`md2map.sh` is a bash script written by James Holton that uses CCP4 programs (*http://www.ccp4.ac.uk/*) to calculate the average electron density from a crystal simulation. `md2map` thus requires the user to have CCP4 installed. It works by calculating the electron density from each simulation trajectory frame (in PDB format), then summing all the electron densities and averaging. Like the other scripts in XtalAnalyze, `md2map.sh` is run by copying the script into a working directory of the user's choosing, modyfing a set of variables in the header of the script and executing on the command line. Running `md2map.sh` usually consists of three steps.

1. Prepare the simulation trajectory to obtain the average electron density map. This trajectory of the supercell should be mass-centered. Usually one can use the *fit.nc* trajectory produced by a run of `XtalAnalyze.sh`.

2. Prepare a set of PDB format files, one for each frame to be used in calculating the average electron density. For this the *MakePdb4Map.py* script in the Analysis directory can be used. Copy this script into your working directory and set the following variables in the header before executing:

   SC_TRAJ - the trajectory from Step 1 above

   SC_TOPO - the corresponding topology

   PDB_FILE - original PDB file of the asymmetric unit which must have the CRYST1 record

   frames - the total number of frames to use

   startframe - the first frame to use

   offsetframe - the number of frames to skip between used frames. To use every frame of the trajectory, set this to 1.

   `MakePdb4Map.py` creates a new directory named *PDBData* containing the PDB format files for each of the frames to be used in calculating the average electron density.

3. Copy `md2map.sh` into the working directory, set variables in the header and execute. The variables to set are:

   sim_pdb_dir - directory where the PDB format files of all the trajectory frames to be used in the calculation are located; usually this is the PDBData directory produced in Step 2 above.

   cell - lengths and angles of the experimental unit cell obtained from CRYST1 record in the original pdb file (separated by spaces, in parentheses)

   MD_mult - the propagations used to create the simulation supercell (in parentheses, separated by spaces)

   SG - the space group symbol used by CCP4 (eg. "P1", "P212121", "P31",etc).

   GRID - grid mesh to use for creating the electron density map. Corresponds to the grid command in SFALL (http://www.ccp4.ac.uk/html/sfall.html#grid). Usually, a grid size corresponding to a grid spacing of 0.3-05 Åis works best. Note that the crystal space group may place special restrictions on the the grid sizes that can be used. For more information consult the SFALL documentation cited above.

   B - the B-factor applied to avoid a singularity when dealing with single point atoms. Usually the default of 15 is used. For more information see Janowski et al.[629].

   reso - resolution limit for calculating the structure factors in the output mtz file. Usually this is set to something slightly higher than the experimental map so that the full set of structure factors corresponding to the experimental data set can later be obtained using *filter_mtz.py.*

   align - An alternative method to aligning the supercell snapshots by center of mass is to align by convoluting the simulation electron density with the experimental model electron density, finding the largest peak, and using that peak's location as the required translation to align the simulation map to the experimental. This option is provided for exprimental use or if for some reason the user does not have a center of mass aligned trajectory. To use this method set align to 1. Otherwise leave 0.

exp_pdb	- experimental asymmetric unit model (usually asu.pdb produced by XtalAnalyze.sh). Only used if align=1.

Running `md2map.sh` will produce *md_avg.map* (CCP4 format map file of the average electron density), *md_avg.mtz* (mtz format file from the structure factors and phases of the average electron density map) and a *maps* directory containing an electron density map of each frame from PDBData.

34. SAXS

34.1. Introduction and theory

Small angle X-ray scattering (SAXS) is a solution based technique that is conventionally used to probe the shape and structure of (bio)molecules. It has long been recognized that the solvent shell around the molecule significantly impacts the shape of the measured SAXS profile. Experimentally, X-ray scattering on biomolecules compare the scattering intensity from the sample of interest to a "blank" with just solvent present, and report the difference, or "excess" intensity:

$$I(\mathbf{q}) = \left\langle |A(\mathbf{q})|^2 \right\rangle_t - \left\langle |B(\mathbf{q})|^2 \right\rangle_t$$

where the $\langle\rangle_t$ bracket indicates the intensities are averaged over the measurement time and volume. $A(\mathbf{q})$ and $B(\mathbf{q})$ are Fourier transforms of the scattering amplitudes for the sample and blank, respectively:

$$\left\langle |A(\mathbf{q})|^2 \right\rangle = \int \langle \tilde{A}(\mathbf{r}) \tilde{A}(\mathbf{r}') \rangle e^{-i\mathbf{q}\cdot(\mathbf{r}-\mathbf{r}')} d\mathbf{r} d\mathbf{r}'$$

with $\tilde{A}(\mathbf{r})$ is the electron density in the system. It has been shown that the total intensity can be approximately (though usefully) rewritten as:[630, 631]

$$I(\mathbf{q}) = [\langle A_1(\mathbf{q}) \rangle - \langle B_1(\mathbf{q}) \rangle]^2 + \left[\left\langle |A_1(\mathbf{q})|^2 \right\rangle - |\langle A_1(\mathbf{q}) \rangle|^2 \right] - \left[\left\langle |B_1(\mathbf{q})|^2 \right\rangle - |\langle B_1(\mathbf{q}) \rangle|^2 \right] \tag{34.1}$$

where $A_1(\mathbf{q})$ and $B_1(\mathbf{q})$ are Fourier transforms for the sample and blank but here only considering regions where there is excess/deficit electron density relative to the bulk value. In RISM, the second and the third terms vanish, leading to:

$$I(\mathbf{q}) = [\langle A_1(\mathbf{q}) \rangle - \langle B_1(\mathbf{q}) \rangle]^2 \tag{34.2}$$

There are now two SAXS programs in Amber: `saxs_rism` for calculating SAXS from distribution function of solvent in grid format (dx files) from 3D-RISM, another one is `saxs_md` which takes input as two sets of coordinates extracted from snapshots of “sample” and “blank” MD simulations (the “sample” MD contains the biomolecule plus water and ions, while the “blank” MD only has pure water + salt).

34.1.1. saxs_rism

Intensity is calculated based on eq. 34.2, neglecting the time-correlation of solvent density. The total excess amplitude is calculated by summing up amplitudes from the biomolecule and the solvent (including ions):

$$A_1(\mathbf{q}) - B_1(\mathbf{q}) = F(\mathbf{q}) = F_{solu}(\mathbf{q}) + F_{grid}(\mathbf{q})$$

where the solute form factor is $F_{solu}(\mathbf{q}) = \sum_j f_j(q) \exp\left(-\frac{B_j q^2}{16\pi^2}\right) \exp\left(-i\mathbf{q}.\mathbf{r_j}\right)$ (with $f_j(q)$ is the atomic scattering factor and B_j is the B-factor) and the contribution from the solvent is $F_{grid}(\mathbf{q}) = \sum_j^{N_{grid}} f_j(\mathbf{q}) \exp\left(-i\mathbf{q}.\mathbf{r_j}\right)$.

The angle averaging is then performed by using Lebedev quadrature to obtain the total intensity:

$$I(q) = \frac{1}{4\pi} \int I(\mathbf{q})\, d\Omega$$

This approach was shown valid up to angles corresponding to $q \simeq 1.5$ Å^{-1}. (For more details, see [631]).

34.1.2. saxs_md

The intensity is calculated based on eq. 34.1, which can be rewritten as:[630]

$$I(\mathbf{q}) = \left|a(\mathbf{q}) - b(\mathbf{q})\right|^2 + \frac{1}{N}\sum_i \left|A_1^{(i)}(\mathbf{q}) - a(\mathbf{q})\right|^2 - \frac{N'+1}{N'(N'-1)}\sum_j \left|B_1^{(j)}(\mathbf{q}) - b(\mathbf{q})\right|^2$$

where $A_1^{(i)}(\mathbf{q})$ and $B_1^{(j)}(\mathbf{q})$ are the scattering amplitudes of the each snapshot from the “sample” and “blank”, respectively, and are computed by:

$$A_1(\mathbf{q}) = \sum_n f_n(\mathbf{q})\, e^{-i\mathbf{q}.\mathbf{r_n}}$$

$a(\mathbf{q})$ and $b(\mathbf{q})$ are the averaged amplitudes for the total N and N’ snapshots, respectively (with each weight w_i)

$$a(\mathbf{q}) = \frac{\sum_i^N w_i A_1^{(i)}(\mathbf{q})}{\sum_i^N w_i}$$

$$b(\mathbf{q}) = \frac{\sum_j^{N'} w_j B_1^{(j)}(\mathbf{q})}{\sum_j^{N'} w_j}$$

The angle averaging is then performed by Lebedev quadrature, as in `saxs_rism`.

$$I(q) = \frac{1}{4\pi}\int I(\mathbf{q})\, d\Omega$$

34.2. Usage

34.2.1. saxs_rism

The program requires solvent distribution in dx format (as output of 3D-RISM) and a pdb file of the biomolecule to compute SAXS signal. If run without input, `saxs_rism` prints the usage and default settings for all parameters.

`--grid_dir`	Location of the folder where all the 3D-RISM outputs found. All files in this folder starting with **guv** will be considered by the program. Atom or ion names must be present in the file name in order for the program to recognize. Currently supporting O, H1 (for water), Li$^+$, Na$^+$, K$^+$, Rb$^+$, Cs$^+$, Mg^{2+}, Sr^{2+}, F$^-$, Cl$^-$, Br$^-$, I$^-$. For example these file names are valid: guv.Cl-.dx, guvfileRb+, guvO. The following file names are NOT valid: abc.O.dx, guvNa.dx, guv.H.dx
`--solute`	pdb file of the solute. Currently only supporting the following atoms: H, O, C, N, P, S, Fe
`--conc_ion`	concentration of salt [mol.l^{-1}]. This is the concentration of the cation. Concentration of the anion will be automatically computed (2x in Mg^{2+} and Sr^{2+} cases)
`--conc_wat`	concentration of water [mol.l^{-1}]. Default is 55.34
`--qcut`	momentum transfer q cutoff [Å^{-1}]. Default is 0.5
`--dq`	q spacing [Å^{-1}]. Default is 0.01
`--cutoff`	real space cutoff [Å]. Only considering grid points within cutoff distance to the nearest solute atom. Default is 20
`--off_cutoff`	using all grid points for calculating SAXS, ignoring cutoff value
`--expli`	flag for using explicit H atoms in pdb file to calculate intensity. Default is to merge H atoms into heavier atoms
`--anom_f`	f’ for anomalous scattering. Currently only applied to Rb$^+$, Sr^{2+} or Br$^-$ grids. Default is 0

`--decomp`	flag for decomposing total intensity into site contributions (usually this leads to 2-5x in computational time)
`--exper`	provide the experimental data to read q from. This will override `dq` and `qcut`
`--exclV`	flag for merging those contribution of the grid points inside the excluded volume of the solute into the solute
`--phase`	turn on this flag will output the phase and error analysis
`--tight`	flag for using tighter convergence criteria for Lebedev quadrature (also leads to more time)
`--bfactor`	flag for using B factor (Debye–Waller factor) in the PDB file to compute intensity
`--output`	output file
`--ncpus`	(need to compile with OpenMP to use this flag) specify the number of threads used. Default is to use all available threads

Example

The following example first run 3D-RISM to calculate the distribution function of water around lysozyme (lys.pdb). The output (guv.O.dx and guv.H1.dx) will then be used to compute SAXS intensity

- Run 3D-RISM to obtain the distribution function around the solute

```
$AMBERHOME/bin/rism3d.snglpnt --pdb lys.pdb --prmtop prmtop --xvv rism.xvv --guv guv
```

- Run saxs_rism

```
$AMBERHOME/bin/saxs_rism --grid_dir . --solute lys.pdb --expli --decomp \
       --bfactor --output saxs.out
```

34.2.2. saxs_md

The program requires two sets of coordinates (both in PDB formats) of the “sample” (biomolecule + solvent) and “blank” (pure solvent) systems. Each snapshot starts with “MODEL”, following by “ATOM” or “HETATM” and ends with “ENDMDL” or “END” (for the last snapshot). These pdb files can be generated directly from the trajectory by using ptraj/cpptraj as following:

```
parm prmtop
trajin md.nc
autoimage
trajout rep.pdb pdb
```

Additionally, you can assign the weight for each snapshot by using “WEIGHT”. This is useful if you want to use only representative snapshots for SAXS calculation. For example, the following is a valid pdb file which assign a weight of 342 for the first snapshot and 148 for the second.

```
MODEL     0
WEIGHT    342
ATOM      1 HO5′ DG5     1      14.902  29.822  29.924  1.00  0.00           H
ATOM      2  O5′ DG5     1      15.380  29.001  30.064  1.00  0.00           O
...
ATOM  72772  H1  WAT  3867      40.377  65.382  83.718  1.00  0.00           H
ATOM  72773  H2  WAT  3867      40.942  64.499  82.466  1.00  0.00           H
ENDMDL
```

```
MODEL    1
WEIGHT   148
ATOM     3  C5' DG5    1      16.744  29.215  29.653  1.00  0.00           C
ATOM     4  H5' DG5    1      16.808  29.389  28.579  1.00  0.00           H
....
END
```

If run without input, `saxs_md` prints the usage and default settings for all parameters.

`--system`	pdb file for the solute system
`--solvent`	pdb file for the solvent
`--qcut`	momentum transfer q cutoff [Å^{-1}]. Default is 1.0
`--dq`	q spacing [Å^{-1}]. Default is 0.01
`--cutoff`	distance cutoff to the solute, keep only waters and ions within cutoff distance from the nearest solute atom. Default is 5.0
`--tight`	flag for using tighter convergence criteria for Lebedev quadrature (leads to more computational time)
`--anom_f`	f' for anomalous scattering. Currently only applied to Rb^+, Sr^{2+} or Br^-. Default is 0
`--expli`	flag for using explicit H atoms in pdb files to calculate SAXS. Default is to merge H atoms into heavier atoms.
`--output`	output file
`--ncpus`	(need to compile with OpenMP to use this flag) specify the number of threads used. Default is to use all available threads

Example

The following example use two pdb files (sample.pdb and solvent.pdb) to compute SAXS.

```
$AMBERHOME/bin/saxs_md --system sample.pdb --solvent solvent.pdb \
          --cutoff 10 --expli --output saxs.out
```

Part VI.

NAB and AmberLite

35. NAB: Introduction

Nucleic acid builder (nab) is a high-level language that facilitates manipulations of macromolecules and their fragments. nab uses a C-like syntax for variables, expressions and control structures (if, for, while) and has extensions for operating on molecules (new types and a large number of builtins for providing the necessary operations). We expect nab to be useful in model building and coordinate manipulation of proteins and nucleic acids, ranging in size from fairly small systems to the largest systems for which an atomic level of description makes good computational sense. As a programming language, it is not a solution or program in itself, but rather provides an environment that eases many of the bookkeeping tasks involved in writing programs that manipulate three-dimensional structural models.

The current implementation incorporates the following main features:

1. Objects such as points, atoms, residues, strands and molecules can be referenced and manipulated as named objects. The internal manipulations involved in operations like merging several strands into a single molecule are carried out automatically; in most cases the programmer need not be concerned about the internal data structures involved.

2. Rigid body transformations of molecules or parts of molecules can be specified with a fairly high-level set of routines. This functionality includes rotations and translations about particular axis systems, least-squares atomic superposition, and manipulations of coordinate frames that can be attached to particular atomic fragments.

3. Additional coordinate manipulation is achieved by a tight interface to distance geometry methods. This allows allows relationships that can be defined in terms of internal distance constraints to be realized in three-dimensional structural models. nab includes subroutines to manipulate distance bounds in a convenient fashion, in order to carry out tasks such as working with fragments within a molecule or establishing bounds based on model structures.

4. Force field calculations (*e.g.* molecular dynamics and minimization) can be carried out with an implementation of the AMBER force field. This works in both three and four dimensions, but periodic simulations are not (yet) supported. However, the generalized Born models implemented in Amber are also implemented here, which allows many interesting simulations to be carried out without requiring periodic boundary conditions. The force field can be used to carry out minimization, molecular dynamics, or normal mode calculations. Conformational searching and docking can be carried out using a "low-mode" (LMOD) procedure that performs sampling exploring the potential energy surface along low-frequency vibrational directions.

5. nab also implements a form of regular expressions that we call *atom regular expressions*, which provide a uniform and convenient method for working on parts of molecules.

6. Many of the general programming features of the *awk* language have been incorporated in nab. These include regular expression pattern matching, *hashedarrays* (i.e., arrays with strings as indices), the splitting of strings into fields, and simplified string manipulations.

7. There are built-in procedures for linking nab routines to other routines written in C or Fortran, including access to most library routines normally available in system math libraries.

Our hope is that nab will serve to formalize the step-by-step process that is used to build complex model structures, and will facilitate the management and use of higher level symbolic constraints. Writing a program to create a structure forces more of the model's assumptions to be explicit in the program itself. And an nab description can serve as a way to show a model's salient features, much like helical parameters are used to characterize duplexes.

This chapters introduces the language through a series of sample programs, and illustrate the programming interfaces provided. The examples are chosen not only to show the syntax of the language, but also to illustrate potential approaches to the construction of some unusual nucleic acids, including DNA double- and triple-helices, RNA pseudoknots, four-arm junctions, and DNA-protein interactions. Subsequent chapters give a more formal and careful description of the requirements of the language itself.

The basic literature reference for the code is T. Macke and D.A. Case. Modeling unusual nucleic acid structures. In *Molecular Modeling of Nucleic Acids*, N.B. Leontes and J. SantaLucia, Jr., eds. (Washington, DC: American Chemical Society, 1998), pp. 379-393. Users are requested to include this citation in papers that make use of NAB.

The authors thank Jarrod Smith, Garry Gippert, Paul Beroza, Walter Chazin, Doree Sitkoff and Vickie Tsui for advice and encouragement. Special thanks to Neill White (who helped in updating documentation, in preparing the distance geometry database, and in testing and porting portions of the code), and to Will Briggs (who wrote the fiber-diffraction routines). Thanks also to Chris Putnam and M.L. Dodson for bug reports.

35.1. Background

Using a computer language to model polynucleotides follows logically from the fundamental nature of nucleic acids, which can be described as “conflicted” or “contradictory” molecules. Each repeating unit contains seven rotatable bonds (creating a very flexible backbone), but also contains a rigid, planar base which can participate in a limited number of regular interactions, such as base pairing and stacking. The result of these opposing tendencies is a family of molecules that have the potential to adopt a virtually unlimited number of conformations, yet have very strong preferences for regular helical structures and for certain types of loops.

The controlled flexibility of nucleic acids makes them difficult to model. On one hand, the limited range of regular interactions for the bases permits the use of simplified and more abstract geometric representations. The most common of these is the replacement of each base by a plane, reducing the representation of a molecule to the set of transformations that relate the planes to each other. On the other hand, the flexible backbone makes it likely that there are entire families of nucleic acid structures that satisfy the constraints of any particular modeling problem. Families of structures must be created and compared to the model’s constraints. From this we can see that modeling nucleic acids involves not just chemical knowledge but also three processes-abstraction, iteration and testing-that are the basis of programming.

Molecular computation languages are not a new idea. Here we briefly describe some past approaches to nucleic acid modeling, to provide a context for nab.

35.1.1. Conformation build-up procedures

MC-SYM[632–634] is a high level molecular description language used to describe single stranded RNA molecules in terms of functional constraints. It then uses those constraints to generate structures that are consistent with that description. MC-SYM structures are created from a small library of conformers for each of the four nucleotides, along with transformation matrices for each base. Building up conformers from these starting blocks can quickly generate a very large tree of structures. The key to MC-SYM’s success is its ability to prune this tree, and the user has considerable flexibility in designing this pruning process.

In a related approach, Erie *et al.*[635] used a Monte-Carlo build-up procedure based on sets of low energy dinucleotide conformers to construct longer low energy single stranded sequences that would be suitable for incorporation into larger structures. Sets of low energy dinucleotide conformers were created by selecting one value from each of the sterically allowed ranges for the six backbone torsion angles and χ. Instead of an exhaustive build- up search over a small set of conformers, this method samples a much larger region of conformational space by randomly combining members of a larger set of initial conformers. Unlike strict build-up procedures, any member of the initial set is allowed to follow any other member, even if their corresponding torsion angles do not exactly match, a concession to the extreme flexibility of the nucleic acid backbone. A key feature determined the probabilities of the initial conformers so that the probability of each created structure accurately reflected its energy.

Tung and Carter[636, 637] have used a reduced coordinate system in the NAMOT (nucleic acid modeling tool) program to rotation matrices that build up nucleic acids from simplified descriptions. Special procedures allow base-pairs to be preserved during deformations. This procedure allows simple algorithmic descriptions to be constructed for non-regular structures like intercalation sites, hairpins, pseudoknots and bent helices.

35.1.2. Base-first strategies

An alternative approach that works well for some problems is the "base-first" strategy, which lays out the bases in desired locations, and attempts to find conformations of the sugar-phosphate backbone to connect them. Rigid-body transformations often provide a good way to place the bases. One solution to the backbone problem would be to determine the relationship between the helicoidal parameters of the bases and the associated backbone/sugar torsions. Work along these lines suggests that the relationship is complicated and non-linear.[638] However, considerable simplification can be achieved if instead of using the complete relationship between all the helicoidal parameters and the entire backbone, the problem is limited to describing the relationship between the helicoidal parameters and the backbone/sugar torsion angles of single nucleotides and then using this information to drive a constraint minimizer that tries to connect adjacent nucleotides. This is the approach used in JUMNA,[639] which decomposes the problem of building a model nucleic acid structure into the constraint satisfaction problem of connecting adjacent flexible nucleotides. The sequence is decomposed into 3'-nucleotide monophosphates. Each nucleotide has as independent variables its six helicoidal parameters, its glycosidic torsion angle, three sugar angles, two sugar torsions and two backbone torsions. JUMNA seeks to adjust these independent variables to satisfy the constraints involving sugar ring and backbone closure.

Even constructing the base locations can be a non-trivial modeling task, especially for non-standard structures. Recognizing that coordinate frames should be chosen to provide a simple description of the transformations to be used, Gabarro-Arpa *et al.*[640] devised “Object Command Language” (OCL), a small computer language that is used to associate parts of molecules called objects, with arbitrary coordinate frames defined by sets of their atoms or numerical points. OCL can “link” objects, allowing other objects’ positions and orientations to be described in the frame of some reference object. Information describing these frames and links is written out and used by the program MORCAD[641] which does the actual object transformations.

OCL contains several elements of a molecular modeling language. Users can create and operate on sets of atoms called objects. Objects are built by naming their component atoms and to simplify creation of larger objects, expressions, IF statements, an iterated FOR loop and limited I/O are provided. Another nice feature is the equivalence between a literal 3-D point and the position represented by an atom’s name. OCL includes numerous built-in functions on 3-vectors like the dot and cross products as well as specialized molecular modeling functions like creating a vector that is normal to an object. However, OCL is limited because these language elements can only be assembled into functions that define coordinate frames for molecules that will be operated on by MORCAD. Functions producing values of other data types and stand-alone OCL programs are not possible.

35.2. Methods for structure creation

As a structure-generating tool, nab provides three methods for building models. They are rigid-body transformations, metric matrix distance geometry, and molecular mechanics. The first two methods are good initial methods, but almost always create structures with some distortion that must be removed. On the other hand, molecular mechanics is a poor initial method but very good at refinement. Thus the three methods work well together.

35.2.1. Rigid-body transformations

Rigid-body transformations create model structures by applying coordinate transformations to members of a set of standard residues to move them to new positions and orientations where they are incorporated into the growing model structure. The method is especially suited to helical nucleic acid molecules with their highly regular structures. It is less satisfactory for more irregular structures where internal rearrangement is required to remove bad covalent or non-bonded geometry, or where it may not be obvious how to place the bases. Details are given in Chap. 37.

nab uses the matrix type to hold a 4×4 transformation matrix. Transformations are applied to residues and molecules to move them into new orientations or positions. nab does *not* require that transformations applied to parts of residues or molecules be chemically valid. It simply transforms the coordinates of the selected atoms leaving it to the user to correct (or ignore) any chemically incorrect geometry caused by the transformation.

Every nab molecule includes a frame, or “handle” that can be used to position two molecules in a generalization of superimposition. Traditionally, when a molecule is superimposed on a reference molecule, the user first forms a correspondence between a set of atoms in the first molecule and another set of atoms in the reference molecule. The superimposition algorithm then determines the transformation that will minimize the rmsd between corresponding atoms. Because superimposition is based on actual atom positions, it requires that the two molecules have a common substructure, and it can only place one molecule on top of another and not at an arbitrary point in space.

The nab frame is a way around these limitations. A frame is composed of three orthonormal vectors originally aligned along the axes of a right handed coordinate frame centered on the origin. nab provides two builtin functions setframe() and setframep() that are used to reposition this frame based on vectors defined by atom expressions or arbitrary 3-D points, respectively. To position two molecules via their frames, the user moves the frames so that when they are superimposed via the nab builtin alignframe(), the two molecules have the desired orientation. This is a generalization of the methods described above for OCL.

35.2.2. Distance geometry

nab’s second initial structure-creation method is *metric matrix distance geometry,[642, 643]* which can be a very powerful method of creating initial structures. It has two main strengths. First, since it uses internal coordinates, the initial position of atoms about which nothing is known may be left unspecified. This has the effect that distance geometry models use only the information the modeler considers valid. No assumptions are required concerning the positions of unspecified atoms. The second advantage is that much structural information is in the form of distances. These include constraints from NMR or fluorescence energy transfer experiments, implied propinquities from chemical probing and footprinting, and tertiary interactions inferred from sequence analysis. Distance geometry provides a way to formally incorporate this information, or other assumptions, into the model-building process. Details are given in Chap. 38.

Distance geometry converts a molecule represented as a set of interatomic distances into a 3-D structure. nab has several builtin functions that are used together to provide metric matrix distance geometry. A bounds object contains the molecule’s interatomic distance bounds matrix and a list of its chiral centers and their volumes. The function newbounds() creates a bounds object containing a distance bounds matrix containing initial upper and lower bounds for every pair of atoms, and a list of the molecule’s chiral centers and their volumes. Distance bounds for pairs of atoms involving only a single residue are derived from that residue’s coordinates. The 1,2 and 1,3 distance bounds are set to the actual distance between the atoms. The 1,4 distance lower bound is set to the larger of the sum of the two atoms van der Waals radii or their *syn* (torsion angle = 0o) distance, and the upper bound is set to their *anti* (torsion angle = 180o) distance. newbounds() also initializes the list of the molecule’s chiral centers. Each chiral center is an ordered list of four atoms and the volume of the tetrahedron those four atoms enclose. Each entry in a nab residue library contains a list of the chiral centers composed entirely of atoms in that residue.

Once a bounds object has been initialized, the modeler can use functions to tighten, loosen or set other distance bounds and chiralities that correspond to experimental measurements or parts of the model’s hypothesis. The functions andbounds() and orbounds() allow logical manipulation of bounds. setbounds_from_db() Allows distance information from a model structure or a database to be incorporated into a part of the current molecule’s bounds object, facilitating transfer of information between partially-built structures.

These primitive functions can be incorporated into higher-level routines. For example the functions stack() and watsoncrick() set the bounds between the two specified bases to what they would be if they were stacked in a strand or base-paired in a standard Watson/Crick duplex, with ranges of allowed distances derived from an analysis of structures in the Nucleic Acid Database.

After all experimental and model constraints have been entered into the bounds object, the function tsmooth() applies “triangle smoothing” to pull in the large upper bounds, since the maximum distance between two atoms can not exceed the sum of the upper bounds of the shortest path between them. Random pairwise metrization[644] can also be used to help ensure consistency of the bounds and to improve the sampling of conformational space. The function embed() finally takes the smoothed bounds and converts them into a 3-D object. The newly embedded coordinates are subject to conjugate gradient refinement against the distance and chirality information contained in bounds. The call to embed() is usually placed in a loop to explore the diversity of the structures the bounds represent.

35.2.3. Molecular mechanics

The final structure creation method that nab offers is *molecular mechanics*. This includes both energy minimization and molecular dynamics - simulated annealing. Since this method requires a good estimate of the initial position of every atom in a structure, it is not suitable for creating initial structures. However, given a reasonable initial structure, it can be used to remove bad initial geometry and to explore the conformational space around the initial structure. This makes it a good method for refining structures created either by rigid body transformations or distance geometry. nab has its own 3-D/4-D molecular mechanics package that implements several AMBER force fields and reads AMBER parameter and topology files. Solvation effects can also be modelled with generalized Born continuum models. Details are given in Chap. 39.

Our hope is that nab will serve to formalize the step-by-step process that is used to build complex model structures. It will facilitate the management and use of higher level symbolic constraints. Writing a program to create a structure forces one to make explicit more of the model's assumptions in the program itself. And an nab description can serve as a way to exhibit a model's salient features, much like helical parameters are used to characterize duplexes. So far, nab has been used to construct models for synthetic Holliday junctions,[645] calcyclin dimers,[646] HMG-protein/DNA complexes,[647] active sites of Rieske iron-sulfur proteins,[648] and supercoiled DNA.[649] The Examples chapter below provides a number of other sample applications.

35.3. Compiling nab Programs

Compiling nab programs is very similar to compiling other high-level language programs, such as C and Fortran. The command line syntax is

```
nab [-O] [-c] [-v] [-noassert] [-nodebug] [-o file] [-Dstring] file(s)
```

where

```
-O optimizes the object code
-c suppresses the linking stage with ld and produces a .o file
-v verbosely reports on the compile process
-noassert causes the compiler to ignore assert statements
-nodebug causes the compiler to ignore debug statements
-o file names the output file (deafult is "a.out" on most operating systems)
-Dstring defines string to the C preprocessor
```

Linking Fortran and C object code with nab is accomplished simply by including the source files on the command line with the nab file. For instance, if a nab program *bar.nab* uses a C function defined in the file *foo.c*, compiling and linking nab code would be accomplished by

```
nab -o bar bar.nab foo.c
```

The result is an executable bar file. To run the program, type:

```
./bar <command line options needed go here>
```

35.4. Parallel Execution

The generalized Born energy routines (for both first and second derivatives) include directives that will allow for parallel execution on machines that support this option. Once you have some level of comfort and experience with the single-CPU version, you can enable parallel execution by supplying one of several parallelization options (*-openmp, -mpi* or *-scalapack*) to configure, by re-building the NAB compiler and by recompiling your NAB program.

The *-openmp* option enables parallel execution under OpenMP on shared- memory machines. To enable OpenMP execution, add the *-openmp* option to configure, re-build the NAB compiler and re-compile your NAB program. Then, if you set the OMP_NUM_THREADS environment variable to the number of threads that you wish to perform parallel execution, the Born energy computation will execute in parallel.

The *-mpi* option enables parallel execution under MPI on either clusters or shared-memory machines. To enable MPI execution, add the *-mpi* option to configure and re-build the NAB compiler. You will not need to modify your NAB programs; just execute them with an `mpirun` command.

The *-scalapack* option enables parallel execution under MPI on either clusters or shared-memory machines, and in addition uses the Scalable LAPACK (ScaLAPACK) library for parallel linear algebra computation that is required to calculate the second derivatives of the generalized Born energy, to perform Newton-Raphson minimization or to perform normal mode analysis. For computations that do not involve linear algebra (such as conjugate gradients minimization or molecular dynamics) the *-scalapack* option functions in the same manner as the *-mpi* option. Do not use the *-mpi* and *-scalapack* options simultaneously. Use the *-scalapack* option only when ScaLAPACK has been installed on your cluster or shared-memory machine.

In order that the *-mpi* or *-scalapack* options result in a correct build of the NAB compiler, the configure script must specify linking of the MPI library, or ScaLAPACK and BLACS libraries, as part of that build. These libraries are specified for Sun machines in the solaris_cc section of the configure script. If you want to use MPI or ScaLAPACK on a machine other than a Sun machine, you will need to modify the configure script to link these libraries in a manner analogous to what occurs in the solaris_cc section of the script.

There are three options to specify the manner in which NAB supports linear algebra computation. The *-scalapack* option discussed above specifies ScaLAPACK. The *-perflib* option specifies Sun TM Performance Library TM , a multi-threaded implementation of LAPACK. If neither *-scalapack* nor *-perflib* is specified, then linear algebra computation will be performed by a single CPU using LAPACK. In this last case, the Intel MKL library will be used if the `MKL_HOME` environment variable is set at configure time. Absent that, if a `GOTO` environment variable is found, the GotoBLAS libraries will be used.

The parallel execution capability of NAB was developed primarily on Sun machines, and has also been tested on the SGI Altix platform. But it has been much less widely-used than have other parts of NAB, so you should certainly run some tests with your system to ensure that single-CPU and parallel runs give the same results.

The *$AMBERHOME/benchmarks/nab* directory has a series of timing benchmarks that can be helpful in assessing performance. See the README file there for more information.

35.5. First Examples

This section introduces nab via three simple examples. All nab programs in this user manual are set in Courier, a typewriter style font. The line numbers at the beginning of each line are not parts of the programs but have been added to make it easier to refer to specific program sections.

35.5.1. B-form DNA duplex

One of the goals of nab was that simple models should require simple programs. Here is an nab program that creates a model of a B-form DNA duplex and saves it as a PDB file.

```
1  // Program 1 - Average B-form DNA duplex
2  molecule m;
3
4  m = bdna( "gcgttaacgc" );
5  putpdb( "gcg10.pdb", m );
```

Line 2 is a declaration used to tell the nab compiler that the name m is a molecule variable, something nab programs use to hold structures. Line 4 creates the actual model using the predefined function bdna(). This function's argument is a literal string which represents the sequence of the duplex that is to be created. Here's how bdna() converts this string into a molecule. Each letter stands for one of the four standard bases: a for adenine, c for cytosine, g for guanine and t for thymine. In a standard DNA duplex every adenine is paired with thymine and every cytosine with guanine in an antiparallel double helix. Thus only one strand of the double helix has to be specified. As bdna() reads the string from left to right, it creates one strand from 5' to 3' (5'-gcgttaacgc -3'), automatically creating the other antiparallel strand using Watson/Crick pairing. It uses a uniform helical step of 3.38 Å rise and 36.0o twist. Naturally, nab has other ways to create helical molecules with arbitrary helical parameters and even mismatched base pairs, but if you need some "average" DNA, you should be able to get it without having to specify every detail. The last line uses the nab builtin putpdb() to write the newly created duplex to the file gcg10.pdb.

Program 1 is about the smallest nab program that does any real work. Even so, it contains several elements common to almost all nab programs. The two consecutive forward slashes in line 1 introduce a comment which tells the nab compiler to ignore all characters between them and the end of the line. This particular comment begins in column 1, but that is not required as comments may begin in any column. Line 3 is blank. It serves no purpose other than to visually separate the declaration part from the action part. nab input is free format. Runs of white space characters—spaces, tabs, blank lines and page breaks—act like a single space which is required only to separate reserved words like molecule from identifiers like m. Thus white space can be used to increase readability.

35.5.2. Superimpose two molecules

Here is another simple nab program. It reads two DNA molecules and superimposes them using a rotation matrix made from a correspondence between their C1' atoms.

```
 // Program 2 - Superimpose two DNA duplexes
molecule m, mr;
float r;

m = getpdb( "test.pdb" );
mr = getpdb( "gcg10.pdb" );
superimpose( m, "::C1'", mr, "::C1'" );
putpdb( "test.sup.pdb", m );
rmsd( m, "::C1'", mr, "::C1'", r );
printf( "rmsd = %8.3fn", r );
```

This program uses three variables—two molecules, m and mr and one float, r. An nab declaration can include any number of variables of the same type, but variables of different types must be in separate declarations. The builtin function getpdb() reads two molecules in PDB format from the files test.pdb and gcg10.pdb into the variables m and mr. The superimposition is done with the builtin function superimpose(). The arguments to superimpose() are two molecules and two "atom expressions". nab uses atom expressions as a compact way of specifying sets of atoms. Atom expressions and atom names are discussed in more detail below but for now an atom expression is a pattern that selects one or more of the atoms in a molecule. In this example, they select all atoms with names C1'.

superimpose() uses the two atom expressions to associate the corresponding C1' carbons in the two molecules. It uses these correspondences to create a rotation matrix that when applied to m will minimize the root mean square deviation between the pairs. It applies this matrix to m, "moving" it on to mr. The transformed molecule m is written out to the file test.sup.pdb in PDB format using the builtin function putpdb(). Finally the builtin function rmsd() is used to compute the actual root mean square deviation between corresponding atoms in the two superimposed molecules. It returns the result in r, which is written out using the C-like I/O function printf(). rmsd() also uses two atom expressions to select the corresponding pairs. In this example, they are the same pairs that were used in the superimposition, but any set of pairs would have been acceptable. An example of how this might be used would be to use different subsets of corresponding atoms to compute trial superimpositions and then use rmsd() over all atoms of both molecules to determine which subset did the best job.

35.5.3. Place residues in a standard orientation

This is the last of the introductory examples. It places nucleic acid monomers in an orientation that is useful for building Watson/Crick base pairs. It uses several atom expressions to create a frame or handle attached to an nab

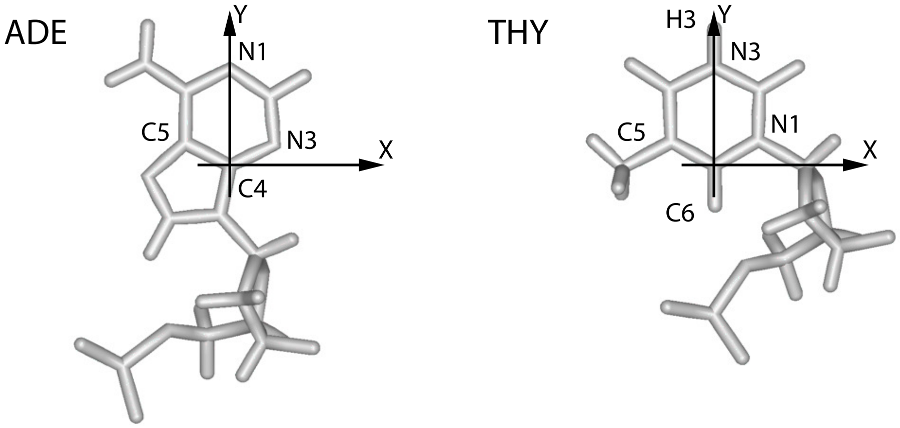


Figure 35.1.: *ADE and THY after execution of Program 3.*

molecule that permits easy movement along important "molecular directions". In a standard Watson/Crick base pair the C4 and N1 atoms of the purine base and the H3, N3 and C6 atoms of the pyrimidine base are colinear. Such a line is obviously an important molecular direction and would make a good coordinate axis. Program 3 aligns these monomers so that this hydrogen bond is along the Y-axis.

```
 1  // Program 3 - orient nucleic acid monomers
 2  molecule m;
 3
 4  m = getpdb( "ADE.pdb" );
 5  setframe( 2, m, // also for GUA
 6            "::C4",
 7            "::C5", "::N3",
 8            "::C4", "::N1" );
 9  alignframe( m, NULL );
10  putpdb( "ADE.std.pdb", m );
11
12  m = getpdb( "THY.pdb" );
13  setframe( 2, m, // also for CYT & URA
14            "::C6",
15            "::C5", "::N1",
16            "::C6", "::N3" );
17  alignframe( m, NULL );
18  putpdb( "THY.std.pdb", m );
```

This program uses only one variable, the molecule m. Execution begins on line 4 where the builtin getpdb() is used to read in the coordinates of an adenine (created elsewhere) from the file ADE.pdb. The nab builtin setframe() creates a coordinate frame for this molecule using vectors defined by some of its atoms as shown in Figure 35.1. The first atom expression (line 6) sets the origin of this coordinate frame to be the coordinates of the C4 atom. The two atom expressions on line 7 set the X direction from the coordinates of the C5 to the coordinates of the N3. The last two atom expressions set the Y direction from the C4 to the N1. The Z-axis is created by the cross product X×Y. Frames are thus like sets of local coordinates that can be attached to molecules and used to facilitate defining transformations; a more complete discussion is given in the section **Frames** below.

nab requires that the coordinate axes of all frames be orthogonal, and while the X and Y axes as specified here are close, they are not quite exact. setframe() uses its first parameter to specify which of the original two axes is to be used as a formal axis. If this parameter is 1, then the specified X axis becomes the formal X axis and Y is recreated from Z×X; if the value is 2, then the specified Y axis becomes the formal Y axis and X is recreated from Y×Z. In this example the specified Y axis is used and X is recreated. The builtin alignframe() transforms the molecule so that the X, Y and Z axes of the newly created coordinate frame point along the standard X, Y and Z directions and that the origin is at (0,0,0). The transformed molecule is written to the file ADE.std.pdb. A similar procedure is performed on a thymine residue with the result that the hydrogen bond between the H3 of thymine and the N1 of adenine in a Watson Crick pair is now along the Y axis of these two residues.

35.6. Molecules, Residues and Atoms

We now turn to a discussion of ways of describing and manipulating molecules. In addition to the general-purpose variable types like float, int and string, nab has three types for working with molecules: molecule, residue and atom. Like their chemical counterparts, nab molecules are composed of residues which are in turn composed of atoms. The residues in an nab molecule are organized into one or more named, ordered lists called strands. Residues in a strand are usually bonded so that the “exiting” atom of residue *i* is connected to the “entering” atom of residue $i+1$. The residues in a strand need not be bonded; however, only residues in the same strand can be bonded.

Each of the three molecular types has a complex internal structure, only some of which is directly accessible at the nab level. Simple elements of these types, like the number of atoms in a molecule or the X coordinate of an atom are accessed via attributes—a suffix attached to a molecule, residue or atom variable. Attributes behave almost like int, float and string variables; the only exception being that some attributes are read only with values that can t be changed. More complex operations on these types such as adding a residue to a molecule or merging two strands into one are handled with builtin functions. A complete list of nab builtin functions and molecule attributes can be found in the nab Language Reference.

35.7. Creating Molecules

The following functions are used to create molecules. Only an overview is given here; more details are in chapter 3.

```
molecule newmolecule();
int addstrand( molecule m, string str );
residue getresidue( string rname, string rlib );
residue transformres( matrix mat, residue res, string aex );
int addresidue( molecule m, string str, residue res );
int connectres( molecule m, string str,
        int rn1, string atm1, int rn2, string atm2 );
int mergestr( molecule m1, string str1, string end1,
        molecule m2, string str2, string end2 );
```

The general strategy for creating molecules with nab is to create a new (empty) molecule then build it one residue at a time. Each residue is fetched from a residue library, transformed to properly position it and added to a growing strand. A template showing this strategy is shown below. mat, m and res are respectively a matrix, molecule and residue variable declared elsewhere. Words in italics indicate general instances of things that would be filled in according to actual application.

```
...
m = newmolecule();
addstrand( m, \fIstr-1\fC );
...
for( ... ){
...
res = getresidue( \fIres-name\fC, \fIres-lib\fC );
res = transformres( mat, res, NULL );
addresidue( m, \fIstr-name\fC, res );
...
11 }
12 ...
```

In line 2, the function newmolecule() creates a molecule and stores it in m. The new molecule is empty—no strands, residues or atoms. Next addstrand() is used to add a strand named *str-1*. Strand names may be up to 255 characters in length and can include any characters except white space. Each strand in a molecule must have a unique name. There is no limit on the number of strands a molecule may have.

The actual structure would be created in the loop on lines 5-11. Each time around the loop, the function getresidue() is used to extract the next residue with the name *res-name* from some residue library *res-lib* and stores it in the residue variable res. Next the function transformres() applies a transformation matrix, held in the matrix variable mat to the residue in res, which places it in the orientation and position it will have in the new molecule. Finally, the function addresidue() appends the transformed residue to the end of the chain of residues in the strand *str-name* of the new molecule.

Residues in each strand are numbered from 1 to *N*, where *N* is the number of residues in that strand. The residue order is the order in which they were inserted with addresidue(). While nab does not require it, nucleic acid chains are usually numbered from 5' to 3' and proteins chains from the N-terminus to the C-terminus. The residues in nucleic acid strands and protein chains are usually bonded with the outgoing end of residue *i* bonded to the incoming end of residue *i*+1. However, as this is not always the case, nab requires the user to explicitly make all interresidue bonds with the builtin connectres().

connectres() makes bonds between two atoms in different residues of the same strand of a molecule. Only residues in the same strand can be bonded. connectres() takes six arguments. They are a molecule, the name of the strand containing the residues to be bonded, and two pairs each of a residue number and the name of an atom in that residue. As an example, this call to connectres(),

```
connectres( m, "sense", i, "O3'", i+1, "P" );
```

connects an atom named "O3'" in residue i to an atom named "P" in residue i+1, creating the phosphate bond that joins two nucleic acid monomers.

The function mergestr() is used to either move or copy the residues in one strand into another strand. Details are provided in chapter 3.

35.8. Residues and Residue Libraries

nab programs build molecules from residues that are parts of residue libraries, which are exactly those distributed with the Amber molecular mechanics programs (see http://ambermd.org/).

nab provides several functions for working with residues. All return a valid residue on success and NULL on failure. The function getres() is written in nab and it source is shown below. transformres() which applies a coordinate transformation to a residue and is discussed under the section **Matrices and Transformations**.

```
residue getresidue( string resname, string reslib );
residue getres( string resname, string reslib );
residue transformres( matrix mat, residue res, string aexp );
```

getresidue() extracts the residue with name resname from the residue library reslib. reslib is the name of a file that either contains the residue information or contains names of other files that contain it. reslib is assumed to be in the directory $AMBERHOME/dat/reslib unless it begins with a slash (/)

A common task of many nab programs is the translation of a string of characters into a structure where each letter in the string represents a residue. Generally, some mapping of one or two character names into actual residue names is required. nab supplies the function getres() that maps the single character names a, c, g, t and u and their 5' and 3' terminal analogues into the residues ADE, CYT, GUA, THY and URA. Here is its source:

```
// getres() - map 1 letter names into 3 letter names
residue getres( string rname, string rlib )
{
   residue res;
5      string map1to3[ hashed ];          // convert residue names
6
7      map1to3["A"] = "ADE";     map1to3["C"] = "CYT";
8      map1to3["G"] = "GUA";     map1to3["T"] = "THY";
9      map1to3["U"] = "URA";
10
11     map1to3["a"] = "ADE";     map1to3["c"] = "CYT";
12     map1to3["g"] = "GUA";     map1to3["t"] = "THY";
13     map1to3["u"] = "URA";
14
15     if( r in map1to3 ) {
16         res = getresidue( map1to3[ r ], rlib );
17     }else{
18         fprintf( stderr, "undefined residue %s\\n", r );
19         exit( 1 );
20     }
21     return( res );
22 };
```

getres() is the first of several nab functions that are discussed in this User Manual. The following explanation will cover not just getres() but will serve as an introduction to user defined nab functions in general.

An nab function is a named group of declarations and statements that is executed as a unit by using the function's name in an expression. nab functions can have special variables called parameters that allow the same function to operate on different data. A function definition begins with a header that describes the function, followed by the function body which is a list of statements and declarations enclosed in braces ({}) and ends with a semicolon. The header to getres() is on line 2 and the body is on lines 3 to 22.

Every nab function header begins with the reserved word that specifies its type, followed by the function's name followed by its parameters (if any) enclosed in parentheses. The parentheses are always required, even if the function does not have parameters. nab functions may return a single value of any of the 10 nab types. nab functions can not return arrays. In symbolic terms every nab function header uses this template:

`type name`**(** ***`parameters?`*** **)**

The parameters (if present) to an nab function are a comma separated list of type variable pairs:

`type1 variable1`**,** ***`type2 variable2`*****, ...**

An nab function may have any number of parameters, including none. Parameters may of any of the 10 nab types, but unlike function values, parameters can be arrays, including *hashed* arrays. The function getres() has two parameters, the two string variables resname and reslib.

Parameters to nab functions are "called by reference" which means that they contain the actual data—not copies of it—that the function was called with. When an nab function parameter is assigned, the actual data in the calling function is changed. The only exception is when an expression is passed as a parameter to an nab function. In this case, the nab compiler evaluates the expression into a temporary (and invisible to the nab programmer) variable and then operates on its contents.

Immediately following the function header is the function body. It is a list of declarations followed by a list of statements enclosed in braces. The list of declarations, the list of statements or both may be empty. getres() has several statements, and a single declaration, the variable res. This variable is a *local variables*. Local variables are defined only when the function is active. If a local variable has the same name as variable defined outside of a it the local variable hides the global one. Local variables can not be parameters.

The statement part of getres() begins on line 6. It consists of several if statements organized into a decision tree. The action of this tree is to translate one of the strings A, , , T, etc., or their lower case equivalents into the corresponding three letter standard nucleic acid residue name and then extract that residue from reslib using the low level residue library function getresidue(). The value returned by getresidue() is stored in the local variable res, except when the input string is not one of those listed above. In that case, getres() writes a message to stderr indicating that it can not translate the input string and sets res to the value NULL. nab uses NULL to represent non-existent values of the types string, file, atom, residue, molecule and bounds. A value of NULL generally means that a variable is uninitialized or that an error occurred in creating it.

A function returns a value by executing a return statement, which is the reserved word return followed by an expression. The return statement evaluates the expression, sets the function value to it and returns control to the point just after the call. The expression is optional but if present the type of the expression must be the same as the type of the function or both must be numeric (int, float). If the expression is missing, the function still returns, but its value is undefined. getres() includes one return statements on line 20. A function also returns with an undefined value when it "runs off the bottom", i.e., executes the last statement before the closing brace and that statement is not a return.

35.9. Atom Names and Atom Expressions

Every atom in an nab molecule has a name. This name is composed of the strand name, the residue *number* and the atom name. As both PDB and off formats require that all atoms in a residue have distinct names, the combination of strand name, residue number and atom name is unique for each atom in a single molecule. Atoms in different molecules, however, may have the same name.

Many nab builtins require the user to specify exactly which atoms are to be covered by the operation. nab does this with special strings called *atom expressions*. An atom expression is a pattern that matches one or more atom names in the specified molecule or residue. An atom expression consists of three parts—a strand part, a residue part and an atom part. The parts are separated by colons (:). Not all three parts are required. An atom expression with no colons consists of only a strand part; it selects *all* atoms in the selected strands. An atom expression with one colon consists of a strand part and a residue part; it selects *all* atoms in the selected residues in the selected strands. An empty part selects all strands, residues or atoms depending on which parts are empty.

nab patterns specify the *entire* string to be matched. For example, the atom pattern C matches only atoms named C , and not those named CA, HC, etc. To match any name that begins with C, use C*, to match any name ending with C, use *C and to match a C in any position use *C*. An atom expression is first parsed into its parts. The strand part is evaluated selecting one or more strands in a molecule. Next the residue part is evaluated. Only residues in selected strands can be selected. Finally the atom part is evaluated and only atoms in selected residues are selected. Here are some typical atom expressions and the atoms they match.

:ADE:	Select all atoms in any residue named ADE. All three parts are present but both the strand and atom parts are empty. The atom expression :ADE selects the same set of atoms.
::C,CA,N	select all atoms with names C, CA or N in all residues in all strands—typically the peptide backbone.
A:1-10,13,URA:C1’	Select atoms named C1’ (the glycosyl-carbons) in residues 1 to 10 and 13 and in any residues named URA in the strand named A.
::C*[^’]	Select all non-sugar carbons. The [^’] is an example of a negated character class. It matches any character in the last position except ’.
::P,O?P,C[3-5]?,O[35]?	The nucleic acid backbone. This P selects phosphorous atoms. The O?P matches phosphate oxygens that have various second letters O1P, O2P or OAP or OBP. The C[3-5]? matches the backbone carbons, C3’, C4’, C5’ or C3*, C4*, C5*. And the O[35]? matches the backbone oxygens O3’, O5’ or O3*, O5*.
:: or :	Select all atoms in the molecule.

An important property of nab atom expressions is that the order in which the strands, residues, and atoms are listed is unimportant. That is, the atom expression "2,1:5,2,3:N1,C1’" is the exact same atom expression as "1,2:3,2,5:C1’,N1". All atom expressions are reordered, internal to nab, in increasing atom number. So, in the above example, the selected atoms will be selected in the following sequence:

```
1:2:N1, 1:2:C1', 1:3:N1, 1:3:C1', 1:5:N1, 1:5:C1', 2:2:N1, 2:2:C1',
2:3:N1, 2:3:C1', 2:5:N1, 2:5:C1'
```

The order in which atoms are selected internal to a specific residue are the order in which they appear in a nab PDB file. As seen in the above example, N1 appears before C1' in all nab nucleic acid residues and PDB files.

35.10. Looping over atoms in molecules

Another thing that many nab programs have to do is visit every atom of a molecule. nab provides a special form of its for-loop for accomplishing this task. These loops have this form:

```
for( a in m ) stmt;
```

a and *m* represent an atom and a molecule variable. The action of the loop is to set *a* to each atom in *m* in this order. The first atom is the first atom of the first residue of the first strand. This is followed by the rest of the atoms of this residue, followed by the atoms of the second residue, etc until all the atoms in the first strand have been visited. The process is then repeated on the second and subsequent strands in *m* until *a* has been set to every atom in *m*. The order of the strands in a molecule is the order in which they were created with addstrand(), the order of the residues in a strand is the order in which they were added with addresidue() and the order of the atoms in a residue is the order in which they are listed in the residue library entry that the residue is based on.

The following program uses two nested for-in loops to compute all the proton-proton distances in a molecule. Distances less than cutoff are written to stdout. The program uses the second argument on the command to hold the cutoff value. The program also uses the =~ operator to compare a character string , in this case an atom name to pattern, specified as a regular expression.

```
 1 // Program 4 - compute H-H distances <= cutoff
 2 molecule    m;
 3 atom        ai, aj;
 4 float       d, cutoff;
 5
 6 cutoff = atof( argv[ 2 ] );
 7 m = getpdb( "gcg10.pdb" );
 8
 9 for( ai in m ){
10     if( ai.atomname !~ "H" )continue;
11     for( aj in m ){
12         if( aj.tatomnum <= ai.tatomnum )continue;
13         if( aj.atomname !~ "H" )continue;
14         if(( d=distp(ai.pos,aj.pos))<=cutoff){
15             printf(
16                 "%3d %-4s %-4s %3d %-4s %-4s %8.3f\\n",
17                 ai.tresnum, ai.resname, ai.atomname,
18                 aj.tresnum, aj.resname, aj.atomname,
19                 d );
20         }
21     }
22 }
```

The molecule is read into m using getpdb(). Two atom variables ai and aj are used to hold the pairs of atoms. The outer loop in lines 9-22 sets ai to each atom in m in the order discussed above. Since this program is only interested in proton-proton distances, if ai is not a proton, all calculations involving that atom can be skipped. The if in line 10 tests to see if ai is a proton. It does so by testing to see if ai's name, available via the atomname attribute doesn't match the regular expression "H". If it doesn't match then the program executes the continue statement also on line 10, which has the effect of advancing the outer loop to its next atom.

>From the section on attributes, ai.atomname behaves like a character string. It can be compared against other character strings or tested to see if it matches a pattern or regular expression. The two operators, =~ and !~ stand

for *match* and *doesn't-match* They also inform the nab compiler that the string on their right hand sides is to be treated like a regular expression. In this case, the regular expression "H" matches any name that contains the letter H, or any proton which is just what is required.

If ai is a proton, then the inner loop from 11-21 is executed. This sets aj to each atom in the same order as the loop in 9. Since distance is reflexive ($dist\ i,j$ = $dist\ j,i$), and the distance between an atom and itself is 0, the inner loop uses the if on line 12 to skip the calculation on aj unless it follows ai in the molecule's atom order. Next the if on line 13 checks to see if aj is a proton, skipping to the next atom if it is not. Finally, the if on line 14 computes the distance between the two protons ai and aj and if it is <= cutoff writes the information out using the C-like I/O function printf().

35.11. Points, Transformations and Frames

nab provides three kinds of geometric objects. They are the types point and matrix and the frame component of a molecule.

35.11.1. Points and Vectors

The nab type point is an object that holds three float values. These values can represent the X, Y and Z coordinates of a point or the components of 3-vector. The individual elements of a point variable are accessed via attributes or suffixes added to the variable name. The three point attributes are "x", "y" and "z". Many nab builtin functions use, return or create point values. Details of operations on points are given in chapter 3.

35.11.2. Matrices and Transformations

nab uses the matrix type to hold a 4×4 transformation matrix. Transformations are applied to residues and molecules to move them into new orientations and/or positions. Unlike a general coordinate transformation, nab transformations can not alter the scale (size) of an object. However, transformations can be applied to a subset of the atoms of a residue or molecule changing its shape. For example, nab would use a transformation to rotate a group of atoms about a bond. nab does *not* require that transformations applied to parts of residues or molecules be chemically valid. It simply transforms the coordinates of the selected atoms leaving it to the user to correct (or ignore) any chemically incorrect geometry caused by the transformation. nab uses the following builtin functions to create and use transformations.

```
matrix newtransform( float dx, float dy, float dz,
       float rx, float ry, float rz );
matrix rot4( molecule m, string tail, string head, float angle );
matrix rot4p( point tail, point head, float angle );
matrix trans4( molecule m, string tail, string head, float distance );
matrix trans4p( point tail, point head, float distance );
residue transformres( matrix mat, residue r, string aex );
int transformmol( matrix mat, molecule m, string aex );
```

nab provides three ways to create a new transformation matrix. The function newtransform() creates a transformation matrix from 3 translations and 3 rotations. It is intended to position objects with respect to the standard X, Y, and Z axes located at (0,0,0). Here is how it works. Imagine two coordinate systems, X, Y, Z and X', Y', Z' that are initially superimposed. newtransform() first rotates the the primed coordinate system about Z by rz degrees, then about Y by ry degrees, then about X by rx degrees. Finally the reoriented primed coordinate system is translated to the point (dx,dy,dz) in the unprimed system. The functions rot4() and rot4p() create a transformation matrix that effects a clockwise rotation by an angle (in degrees) about an axis defined by two points. The points can be specified implicitly by atom expressions applied to a molecule in rot4() or explicitly as points in rot4p(). If an atom expression in rot4() selects more that one atom, the average coordinate of all selected atoms is used as the point's value. (Note that a positive rotation angle here is defined to be clockwise, which is in accord with the IUPAC rules for defining torsional angles in molecules, but is opposite to the convention found in many other branches of mathematics.) Similarly, the functions trans4() and trans4p() create a transformation that effects a translation by a distance along the axis defined by two points. A positive translation is from tail to head.

transformres() applies a transformation to those atoms of res that match the atom expression aex. It returns a *copy* of the input residue with the changed coordinates. The input residue is unchanged. It returns NULL if the new residue could not be created. transformmol() applies a transformation to those atoms of mol that match aex . Unlike transformres(), transformmol() *changes* the coordinates of the input molecule. It returns the number of atoms selected by aex. In both functions, the special atom expression NULL selects all atoms in the input residue or molecule.

35.11.3. Frames

Every nab molecule includes a frame, a handle that allows arbitrary and precise movement of the molecule. This frame is set with the nab builtins setframe() and setframep(). It is initially set to the standard X, Y and Z directions centered at (0,0,0). setframe() creates a coordinate frame from atom expressions that specify the the origin, the X direction and the Y direction. If any atom expression selects more that one atom, the average of the selected atoms' coordinates is used. Z is created from X×Y. Since the initial X and Y directions are unlikely to be orthogonal, the use parameter specifies which of the input X and Y directions is to become the formal X or Y direction. If use is 1, X is chosen and Y is recreated from Z×X. If use is 2, then Y is chosen and X is recreated from Y×Z. setframep() is identical except that the five points defining the frame are explicitly provided.

```
int setframe( int use, molecule mol, string origin,
         string xtail, string xhead,
         string ytail, string yhead );
int setframep( int use, molecule mol, point origin,
         point xtail, point xhead,
         point ytail, point yhead );
int alignframe( molecule mol, molecule mref );
```

alignframe() is similar to superimpose(), but works on the molecules' frames rather than selected sets of their atoms. It transforms mol to superimpose its *frame* on the *frame* of mref. If mref is NULL, alignframe() superimposes the frame of mol on the standard X, Y and Z coordinate system centered at (0,0,0).

Here's how frames and transformations work together to permit precise motion between two molecules. Corresponding frames are defined for two molecules. These frames are based on molecular directions. alignframe() is first used to align the frame of one molecule along with the standard X, Y and Z directions. The molecule is then moved and reoriented via transformations. Because its initial frame was along these molecular directions, the transformations are likely to be along or about the axes. Finally alignframe() is used to realign the transformed molecule on the frame of the fixed molecule.

One use of this method would be the rough placement of a drug into a groove on a DNA molecule to create a starting structure for restrained molecular dynamics. setframe() is used to define a frame for the DNA along the appropriate groove, with its origin at the center of the binding site. A similar frame is defined for the drug. alignframe() first aligns the drug on the standard coordinate system whose axes are now important directions between the DNA and the drug. The drug is transformed and alignframe() realigns the transformed drug on the DNA's frame.

35.12. Creating Watson Crick duplexes

Watson/Crick duplexes are fundamental components of almost all nucleic acid structures and nab provides several functions for use in creating them. They are

```
residue getres( string resname, string reslib );
molecule bdna( string seq );
molecule fd_helix( string helix_type, string seq, string acid_type );
string wc_complement( string seq, string reslib, string natype );
molecule wc_basepair( residue sres, residue ares );
molecule wc_helix( string seq, string rlib, string natype,
        string aseq, string arlib, string anatype, float xoff,
        float incl, float twist, float rise, string opts );
```

All of these functions are written in nab allowing the user to modify or extend them as needed without having to modify the nab compiler.

Note: If you just want to create a regular helical structure with a given sequence, use the "fiber-diffraction" routine fd_helix(), which is discussed in Section 36.13. The methods discussed next are more general, and can be extended to more complicated problems, but they are also much harder to follow and understand. The construction of "unusual" nucleic acids was the original focus of NAB; if you are using NAB for some other purpose (such as running Amber force field calculations) you should probably skip to Chapter 39 at this point.

35.12.1. bdna() and fd_helix()

The function bdna() which was used in the first example converts a string into a Watson/Crick DNA duplex using average DNA helical parameters.

```
// bdna() - create average B-form duplex
molecule bdna( string seq )
{
        molecule m;
        string cseq;
        cseq = wc_complement( seq, "", "dna" );
        m = wc_helix( seq, "", "dna",
                cseq, "", "dna",
                2.25, -4.96, 36.0, 3.38, "s5a5s3a3" );
         return( m );
};
```

bdna() calls wc_helix() to create the molecule. However, wc_helix() requires both strands of the duplex so bdna() calls wc_complement() to create a string that represents the Watson/Crick complement of the sequence contained in its parameter seq. The string *"s5a5s3a3"* replaces both the *s*ense and *a*nti 5' terminal phosphates with hydrogens and adds hydrogens to both the *s*ense and *a*nti 3' terminal O3' oxygens. The finished molecule in m is returned as the function's value. If any errors had occurred in creating m, it would have the value NULL, indicating that bdna() failed.

Note that the simple method used in bdna() for constructing the helix is not very generic, since it assumes that the *internal* geometry of the residues in the (default) library are appropriate for this sort of helix. This is in fact the case for B-DNA, but this method cannot be trivially generalized to other forms of helices. One could create initial models of other helical forms in the way described above, and fix up the internal geometry by subsequent energy minimization. An alternative is to directly use fiber-diffraction models for other types of helices. The fd_helix() routine does this, reading a database of experimental coordinates from fiber diffraction data, and constructing a helix of the appropriate form, with the helix axis along *z*. More details are given in Section 36.13.

35.12.2. wc_complement()

The function wc_complement() takes three strings. The first is a sequence using the standard one letter code, the second is the name of an nab residue library, and the third is the nucleic acid type (RNA or DNA). It returns a string that contains the Watson/Crick complement of the input sequence in the same one letter code. The input string and the returned complement string have opposite directions. If the left end of the input string is the 5' base then the left end of the returned string will be the 3' base. The actual direction of the two strings depends on their use.

```
// wc_complement() - create a string that is the W/C
// complement of the string seq
string wc_complement( string seq, string rlib, string rlt )
//  (note that rlib is unused: included only for backwards compatibility
5  {
6     string acbase, base, wcbase, wcseq;
7     int i, len;
8
9     if( rlt == "dna" )      acbase = "t";
10    else if( rlt == "rna" ) acbase = "u";
11    else{
12       fprintf( stderr,
13         "wc_complement: rlt (%s) is not dna/rna, no W/C comp.", rlt );
14       return( NULL );
15    }
16    len = length( seq );
17    wcseq = NULL;
18    for( i = 1; i <= len; i = i + 1 ){
19        base = substr( seq, i, 1 );
20        if( base == "a" || base == "A" )       wcbase = acbase;
21        else if( base == "c" || base == "C" )  wcbase = "g";
22        else if( base == "g" || base == "G" )  wcbase = "c";
23        else if( base == "t" || base == "T" )  wcbase = "a";
24        else if( base == "u" || base == "U" )  wcbase = "a";
25        else{
26            fprintf( stderr, "wc_complement: unknown base %sn", base );
27            return( NULL );
28        }
29        wcseq = wcseq + wcbase;
30    }
31    return( wcseq );
32 }
```

wc_complement() begins its work in line 9, where the nucleic acid type, as indicated by rlt as DNA or RNA is used to determine the correct complement for an a. The complementary sequence is created in the for loop that begins in line 18 and extends to line 30. The nab builtin substr() is used to extract single characters from the input sequence beginning with with position 1 and working from left to right until entire input sequence has been converted. The if-tree from lines 20 to 28 is used to set the character complementary to the current character, using the previously determined acbase if the input character is an a or A. Any character other than the expected a, c, g, t, u (or A, C, G, T, U) is an error causing wc_complement() to print an error message and return NULL, indicating that it failed. Line 29 shows how nab uses the infix + to concatenate character strings. When the entire string has been complemented, the for loop terminates and the complementary sequence now in wcseq is returned as the function value. Note that if the input sequence is empty, wc_complement() returns NULL, indicating failure.

35.12.3. wc_helix() Overview

wc_helix() generates a uniform helical duplex from a sequence, its complement, two residue libraries and four helical parameters: x-offset, inclination, twist and rise. By using two residue libraries, wc_helix() can generate RNA/DNA heteroduplexes. wc_helix() returns an nab molecule containing two strands. The string seq becomes the "sense" strand and the string aseq becomes the "anti" strand. seq and aseq are required to be complementary although this is not checked. wc_helix() creates the molecule one base pair at a time. seq is read from left to right, aseq is read from right to left and corresponding letters are extracted and converted to residues by getres(). These residues are in turn combined into an idealized Watson/Crick base pair by wc_basepair(). An AT created by wc_basepair() is shown in Figure 2.

A Watson/Crick duplex can be modeled as a set of planes stacked in a helix. The numbers that describe the relationships between the planes and between the planes and the helical axis are called helical parameters. Planes can be defined for each base or base pair. Six numbers (three displacements and three angles) can be defined for every pair of planes; however, helical parameters for nucleic acid bases are restricted to the six numbers describing the the relationship between the two bases in a base pair and the six numbers describing the relationship between

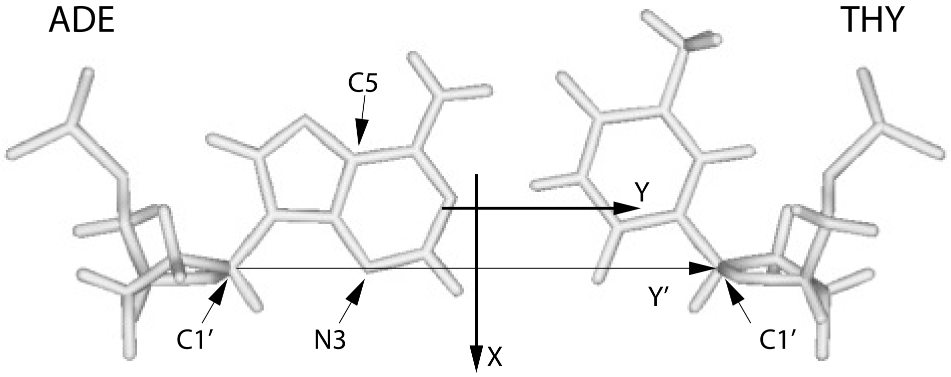


Figure 35.2.: *ADE.THY from wc_basepair().*

adjacent base pairs. A complete description of helical parameters can be found in Dickerson.[650]

wc_helix() uses only four of the 12 helical parameters. It builds its helices from idealized Watson/Crick pairs. These pairs are planar so the three intra base angles are 0. In addition the displacements are displacements from the idealized Watson/Crick geometry and are also 0. The A and the T in Figure 2 are in plane of the page. wc_helix() uses four of the six parameters that relate a base pair to the helical axis. The helices created by wc_helix() have a single axis (the Z axis, not shown) which is at the intersection of the X and Y axes of Figure 2. Now imagine keeping the axes fixed in the plane of the paper and moving the base pair. X-offset is the displacement along the X axis between the Y axis and the line marked Y'. A positive X-offset is toward the arrow on the X-axis. Inclination is the rotation of the base pair about the X axis. A rotation that moves the A above the plane of page and the T below is positive. Twist involves a rotation of the base pair about the Z-axis. A counterclockwise twist is positive. Finally, rise is a displacement along the Z-axis. A positive rise is out of the page toward the reader.

35.12.4. wc_basepair()

The function wc_basepair() takes two residues and assembles them into a two stranded nab molecule containing one base pair. Residue sres is placed in the "sense" strand and residue ares is placed in the "anti" strand. The work begins in line 14 where newmolecule() is used to create an empty molecule stored in m. Two strands, sense and anti are added using addstrand(). In addition, two more molecules are created, m_sense for the sense residue and m_anti for the anti residue. The if-trees in lines 26-61 and 63-83 are used to select residue dependent atoms that will be used to move the base pairs into a convenient orientation for helix generation. The *purine*:C4 and *pyrimidine*:C6 distance which is residue dependent is also set. In line 62, addresidue() adds sres to the strand sense of m_sense. In line 84, addresidue() adds ares to the strand anti of m_anti. Lines 86 and 87 align the molecules containing the sense residue and anti residue so that sres and ares are on top of each other. Line 88 creates a transformation matrix that rotates m_anti (containing ares) 180o about the X-axis. After applying this transformation, the two bases are still occupying the same space but ares is now antiparallel to sres. Line 90 creates a transformation matrix that displaces m_anti and ares along the Y-axis by sep. The properly positioned molecules containing sres and ares are merged into a single molecule, m, completing the base pair. Lines 97-98 move this base pair to a more convenient orientation for helix generation. Initially the base as shown in Figure 35.2 is in the plane of page with origin on the C4 of the A. The calls to setframe() and alignframe() move the base pair so that the origin is at the intersection of the lines marked X and Y'.

```
1  // wc_basepair() - create Watson/Crick base pair
2  #define AT_SEP 8.29
3  #define CG_SEP 8.27
```

```

molecule wc_basepair( residue sres, residue ares )
{
        molecule m, m_sense, m_anti;
        float sep;
        string srname, arname;
        string xtail, xhead;
        string ytail, yhead;
        matrix mat;

        m = newmolecule();
        m_sense = newmolecule();
        m_anti = newmolecule();
        addstrand( m, "sense" );
        addstrand( m, "anti" );
        addstrand( m_sense, "sense" );
        addstrand( m_anti, "anti" );

        srname = getresname( sres );
        arname = getresname( ares );
        ytail = "sense::C1'";
        yhead = "anti::C1'";
        if( ( srname == "ADE" ) || ( srname == "DA" ) ||
            ( srname == "RA" ) || ( srname =~ "[DR]A[35]" ) ){
                sep = AT_SEP;
                xtail = "sense::C5";
                xhead = "sense::N3";
                setframe( 2, m_sense,
                        "::C4", "::C5", "::N3", "::C4", "::N1" );
        }else if( ( srname == "CYT" ) || ( srname =~ "[DR]C[35]*" ) ){
                sep = CG_SEP;
                xtail = "sense::C6";
                xhead = "sense::N1";
                setframe( 2, m_sense,
                        "::C6", "::C5", "::N1", "::C6", "::N3" );
        }else if( ( srname == "GUA" ) || ( srname =~ "[DR]G[35]*" ) ){
                sep = CG_SEP;
                xtail = "sense::C5";
                xhead = "sense::N3";
                setframe( 2, m_sense,
                        "::C4", "::C5", "::N3", "::C4", "::N1" );
        }else if( ( srname == "THY" ) || ( srname =~ "DT[35]*" ) ){
                sep = AT_SEP;
                xtail = "sense::C6";
                xhead = "sense::N1";
                setframe( 2, m_sense,
                        "::C6", "::C5", "::N1", "::C6", "::N3" );
        }else if( ( srname == "URA" ) || ( srname =~ "RU[35]*" ) ){
                sep = AT_SEP;
                xtail = "sense::C6";
                xhead = "sense::N1";
                setframe( 2, m_sense,
                        "::C6", "::C5", "::N1", "::C6", "::N3" );
        }else{
                fprintf( stderr,
                        "wc_basepair : unknown sres %s\\n",srname );
                exit( 1 );
        }
        addresidue( m_sense, "sense", sres );
        if( ( arname == "ADE" ) || ( arname == "DA" ) ||
            ( arname == "RA" ) || ( arname =~ "[DR]A[35]" ) ){
                setframe( 2, m_anti,
                        "::C4", "::C5", "::N3", "::C4", "::N1" );
        }else if( ( arname == "CYT" ) || ( arname =~ "[DR]C[35]*" ) ){
                setframe( 2, m_anti,
                        "::C6", "::C5", "::N1", "::C6", "::N3" );
        }else if( ( arname == "GUA" ) || ( arname =~ "[DR]G[35]*" ) ){
                setframe( 2, m_anti,
                        "::C4", "::C5", "::N3", "::C4", "::N1" );
        }else if( ( arname == "THY" ) || ( arname =~ "DT[35]*" ) ){
                setframe( 2, m_anti,
                        "::C6", "::C5", "::N1", "::C6", "::N3" );
        }else if( ( arname == "URA" ) || ( arname =~ "RU[35]*" ) ){
                setframe( 2, m_anti,
                        "::C6", "::C5", "::N1", "::C6", "::N3" );
        }else{
                fprintf( stderr,
                        "wc_basepair : unknown ares %s\\n",arname );
                exit( 1 );
        }
        addresidue( m_anti, "anti", ares );

        alignframe( m_sense, NULL );
        alignframe( m_anti, NULL );
        mat = newtransform( 0., 0., 0., 180., 0., 0. );
        transformmol( mat, m_anti, NULL );
        mat = newtransform( 0., sep, 0., 0., 0., 0. );
        transformmol( mat, m_anti, NULL );
        mergestr( m, "sense", "last", m_sense, "sense", "first" );
        mergestr( m, "anti", "last", m_anti, "anti", "first" );

        freemolecule( m_sense ); freemolecule( m_anti );

        setframe( 2, m, "::C1'", xtail, xhead, ytail, yhead );
        alignframe( m, NULL );
        return( m );
};
```

35.12.5. wc_helix() Implementation

The function wc_helix() assembles base pairs from wc_basepair() into a helical duplex. It is a fairly complicated function that uses several transformations and shows how mergestr() is used to combine smaller molecules into a larger one. In addition to creating complete duplexes, wc_helix() can also create molecules that contain only one strand of a duplex. Using the special value NULL for either seq or aseq creates a duplex that omits the residues for the NULL sequence. The molecule still contains two strands, sense and anti, but the strand corresponding to the NULL sequence has zero residues. wc_helix() first determines which strands are required, then creates the first base pair, then creates the subsequent base pairs and assembles them into a helix and finally packages the requested strands into the returned molecule.

Lines 20-34 test the input sequences to see which strands are required. The variables has_s and has_a are flags where a value of 1 indicates that seq and/or aseq was requested. If an input sequence is NULL, wc_complement() is used to create it and the appropriate flag is set to 0. The nab builtin setreslibkind() is used to set the nucleic acid type so that the proper residue (DNA or RNA) is extracted from the residue library.

The first base pair is created in lines 42-63. The two letters corresponding the 5' base of seq and the 3' base of aseq are extracted using the nab builtin substr(), converted to residues using getresidue() and assembled into a base pair by wc_basepair(). This base pair is oriented as in Figure 2 with the origin at the intersection of the lines X and Y'. Two transformations are created, xomat for the x-offset and inmat for the inclination and applied to this pair.

Base pairs 2 to slen-1 are created in the for loop in lines 66-87. substr() is used to extract the appropriate letters from seq and aseq which are converted into another base pair by getresidue() and wc_basepair(). Four transformations are applied to these base pairs - two to set the x-offset and the inclination and two more to set the twist and the rise. Next m2, the molecule containing the newly created properly positioned base pair must be bonded to the previously created molecule in m1. Since nab only permits bonds between residues in the same strand, mergestr() must be used to combine the corresponding strands in the two molecules before connectres() can create the bonds.

Because the two strands in a Watson/Crick duplex are antiparallel, adding a base pair to one end requires that one residue be added *after* the *last* residue of one strand and that the other residue added *before* the *first* residue of the other strand. In wc_helix() the sense strand is extended after its last residue and the anti strand is extended before its first residue. The call to mergestr() in line 79 extends the sense strand of m1 with the the residue of the sense strand of m2. The residue of m2 is added after the "last" residue of of the sense strand of m1. The final argument "first" indicates that the residue of m2 are copied in their original order m1:sense:last is followed by m2:sense:first. After the strands have been merged, connectres() makes a bond between the O3' of the next to last residue (i-1) and the P of the last residue (i). The next call to mergestr() works similarly for the residues in the anti strands. The residue in the anti strand of m2 are copied into the the anti strand of m1 *before* the first residue of the anti strand of m1 m2:anti:*last* precedes m1:anti:*first* . After merging connectres() creates a bond between the O3' of the new first residue and the P of the second residue.

Lines 121-130 create the returned molecule m3. If the flag has_s is 1, mergestr() copies the entire sense strand of m1 into the empty sense strand of m3. If the flag has_a is 1, the anti strand is also copied.

```
1   // wc_helix() - create Watson/Crick duplex
2   string  wc_complement();
3   molecule wc_basepair();
4   molecule wc_helix(
5          string seq, string sreslib, string snatype,
6          string aseq, string areslib, string anatype,
7          float xoff, float incl, float twist, float rise,
8          string opts )
9   {
10  molecule m1, m2, m3;
11  matrix xomat, inmat, mat;
12  string arname, srname;
13  string sreslib_use, areslib_use;
14  string loup[ hashed ];
15  residue sres, ares;
16  int     has_s, has_a;
17  int i, slen;
18  float   ttwist, trise;
19
20  has_s = 1; has_a = 1;
21  if( sreslib == "" ) sreslib_use = "all_nucleic94.lib";
22         else sreslib_use = sreslib;
23  if( areslib == "" ) areslib_use = "all_nucleic94.lib";
24         else areslib_use = areslib;
25
26  if( seq == NULL && aseq == NULL ){
27         fprintf( stderr, "wc_helix: no sequence\\n" );
28         return( NULL );
29  }else if( seq == NULL ){
30         seq = wc_complement( aseq, areslib_use, snatype );
```

```
        has_s = 0;
}else if( aseq == NULL ){
        aseq = wc_complement( seq, sreslib_use, anatype );
        has_a = 0;
}

slen = length( seq );
loup["g"] = "G"; loup["a"] = "A";
loup["t"] = "T"; loup["c"] = "C";

//                         handle the first base pair:
setreslibkind( sreslib_use, snatype );
srname = "D" + loup[ substr( seq, 1, 1 ) ];
if( opts =~ "s5" )
    sres = getresidue( srname + "5", sreslib_use );
else if( opts =~ "s3" && slen == 1 )
    sres = getresidue( srname + "3", sreslib_use );
else sres = getresidue( srname, sreslib_use );

setreslibkind( areslib_use, anatype );
arname = "D" + loup[ substr( aseq, 1, 1 ) ];
if( opts =~ "a3" )
    ares = getresidue( arname + "3", areslib_use );
else if( opts =~ "a5" && slen == 1 )
    ares = getresidue( arname + "5", areslib_use );
else ares = getresidue( arname, areslib_use );
m1 = wc_basepair( sres, ares );
freeresidue( sres ); freeresidue( ares );
xomat = newtransform(xoff, 0., 0., 0., 0., 0. );
transformmol( xomat, m1, NULL );
inmat = newtransform( 0., 0., 0., incl, 0., 0.);
transformmol( inmat, m1, NULL );

//                         add in the main portion of the helix:
trise = rise; ttwist = twist;
for( i = 2; i <= slen-1; i = i + 1 ){
        srname = "D" + loup[ substr( seq, i, 1 ) ];
        setreslibkind( sreslib, snatype );
        sres = getresidue( srname, sreslib_use );
        arname = "D" + loup[ substr( aseq, i, 1 ) ];
        setreslibkind( areslib, anatype );
        ares = getresidue( arname, areslib_use );
        m2 = wc_basepair( sres, ares );
        freeresidue( sres ); freeresidue( ares );
        transformmol( xomat, m2, NULL );
        transformmol( inmat, m2, NULL );
        mat = newtransform( 0., 0., trise, 0., 0., ttwist );
        transformmol( mat, m2, NULL );
        mergestr( m1, "sense", "last", m2, "sense", "first" );
        connectres( m1, "sense", i-1, "O3'", i, "P" );
        mergestr( m1, "anti", "first", m2, "anti", "last" );
        connectres( m1, "anti", 1, "O3'", 2, "P" );
        trise = trise + rise;
        ttwist = ttwist + twist;
        freemolecule( m2 );
}

```

```

i = slen;         // add in final residue pair:

if( i > 1 ){
       srname = substr( seq, i, 1 );
       srname = "D" + loup[ substr( seq, i, 1 ) ];
       setreslibkind( sreslib, snatype );
       if( opts =~ "s3"  )
          sres = getres( srname + "3", sreslib_use );
       else
          sres = getres( srname, sreslib_use );
       arname = "D" + loup[ substr( aseq, i, 1 ) ];
       setreslibkind( areslib, anatype );
       if( opts =~ "a5" )
          ares = getres( arname + "5", areslib_use );
       else
          ares = getres( arname, areslib_use );

       m2 = wc_basepair( sres, ares );
       freeresidue( sres ); freeresidue( ares );
       transformmol( xomat, m2, NULL );
       transformmol( inmat, m2, NULL );
       mat = newtransform( 0., 0., trise, 0., 0., ttwist );
       transformmol( mat, m2, NULL );
       mergestr( m1, "sense", "last", m2, "sense", "first" );
       connectres( m1, "sense", i-1, "O3'", i, "P" );
       mergestr( m1, "anti", "first", m2, "anti", "last" );
       connectres( m1, "anti", 1, "O3'", 2, "P" );
       trise = trise + rise;
       ttwist = ttwist + twist;
       freemolecule( m2 );
}

m3 = newmolecule();
addstrand( m3, "sense" );
addstrand( m3, "anti" );
if( has_s )
       mergestr( m3, "sense", "last", m1, "sense", "first" );
if( has_a )
       mergestr( m3, "anti", "last",  m1, "anti", "first" );
freemolecule( m1 );

return( m3 );
};
```

36. NAB: Language Reference

nab is a computer language used to create, modify and describe models of macromolecules, especially those of unusual nucleic acids. The following sections provide a complete description of the nab language. The discussion begins with its lexical elements, continues with sections on expressions, statements and user defined functions and concludes with an explanation of each of nab's builtin functions. Two appendices contain a more detailed and formal description of the lexical and syntactic elements of the language including the actual lex and yacc input used to create the compiler. Two other appendices describe nab's internal data structures and the C code generated to support some of nab's higher level operations.

36.1. Language Elements

An nab program is composed of several basic lexical elements: identifiers, reserved words, literals, operators and special characters. These are discussed in the following sections.

36.1.1. Identifiers

An identifier is a sequence of letters, digits and underscores beginning with a letter. Upper and lower case letters are distinct. Identifiers are limited to 255 characters in length. The underscore (_) is a letter. Identifiers beginning with underscore must be used carefully as they may conflict with operating system names and nab created temporaries. Here are some nab identifiers.

mol i3 twist TWIST Watson_Crick_Base_Pair

36.1.2. Reserved Words

Certain identifiers are reserved words, special symbols used by nab to denote control flow and program structure. Here are the nab reserved words:

allocate	assert	atom	bounds	break
continue	deallocate	debug	delete	dynamic
else	file	for	float	hashed
if	in	int	matrix	molecule
point	residue	return	string	while

36.1.3. Literals

Literals are self defining terms used to introduce constant values into expressions. nab provides three types of literals: integers, floats and character strings. Integer literals are sequences of one or more decimal digits. Float literals are sequences of decimal digits that include a decimal point and/or are followed by an exponent. An exponent is the letter e or E followed by an optional + or - followed by one to three decimal digits. The exponent is interpreted as “times 10 to the power of *exp*” where *exp* is the number following the e or E. All numeric literals are base 10. Here are some integer and float literals:

1 3.14159 5 .234 3.0e7 1E-7

String literals are sequences of characters enclosed in double quotes ("). A double quote is placed into a string literal by preceding it with a backslash (\). A backslash is inserted into a string by preceding it with a backslash. Strings of zero length are permitted.

"" "a string" "string with a \"" "string with a \\"

Non-printing characters are inserted into strings via escape sequences: one to three characters following a backslash. Here are the nab string escapes and their meanings:

\a	Bell (a for audible alarm)
\b	Back space
\f	Form feed (new page)
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\”	Literal double quote
\\	Literal backspace
\ooo	Octal character
\xhh	Hex character (hh is 1 or 2 hex digits

Here are some strings with escapes:

"Molecule\tResidue\tAtom\n"
"\252Real quotes\272"

The second string has octal values, \252, the left double quote, and \272, the right double quote.

36.1.4. Operators

nab uses several additional 1 or 2 character symbols as operators. Operators combine literals and identifiers into expressions.

Operator	Meaning	Precedence	Associates
( )	expression grouping	9	
[ ]	array indexing	9	
.	select attribute	8	
unary −	negation	8	right to left
!	not	8	
^	cross product	6	left to right
@	dot product	6	
*	multiplication	6	left to right
/	division	6	left to right
%	modulus	6	left to right
+	addition, concatenation	5	left to right
binary −	subtraction	5	left to right
<	less than	4	
<=	less than or equal to	4	
==	equal	4	
!=	not equal	4	
>=	greater than or equal to	4	
>	greater than	4	
=~	match	4	
!~	doesn’t match	4	
in	hashed array member *or* atom in molecule	4	
&&	and	3	
||	or	2	
=	assignment	1	right to left

36.1.5. Special Characters

nab uses braces ({}) to group statements into compound statements and statements and declarations into function bodies. The semicolon (;) is used to terminate statements. The comma (,) separates items in parameter lists and declarations. The sharp (#) used in column 1 designates a preprocessor directive, which invokes the standard C preprocessor to provide constants, macros and file inclusion. A # in any other column, except in a comment or a literal string is an error. Two consecutive forward slashes (//) indicate that the rest of the line is a comment which is ignored. All other characters except white space (spaces, tabs, newlines and formfeeds) are illegal except in literal strings and comments.

36.2. Higher-level constructs

36.2.1. Variables

A variable is a name given to a part of memory that is used to hold data. Every nab variable has type which determines how the computer interprets the variable's contents. nab provides 10 data types. They are the numeric types int and float which are translated into the underlying C compiler's int and double respectively.*

The string type is used to hold null (zero byte) terminated (C) character strings. The file type is used to access files (equivalent to C's FILE *). There are three types—atom, residue and molecule for creating and working with molecules. The point type holds three float values which can represent the X, Y and Z coordinates of a point or the components of a 3-vector. The matrix type holds 16 float values in a 4×4 matrix and the bounds type is used to hold distance bounds and other information for use in distance geometry calculations.

nab string variables are mapped into C char * variables which are allocated as needed and freed when possible. However, all of this is invisible at the nab level where strings are atomic objects. The atom, residue, molecule and bounds types become pointers to the appropriate C structs. point and matrix are implemented as float [3] and float [4][4] respectively. Again the nab compiler automatically generates all the C code required to makes these types appear as atomic objects.

Every nab variable must be declared. All declarations for functions or variables in the main block must precede the first executable statement of that block. Also all declarations in a user defined nab function must precede the first executable statement of that function. An nab variable declaration begins with the reserved word that specifies the variable's type followed by a comma separated list of identifiers which become variables of that type. Each declaration ends with a semicolon.

```
int i, j, j;
matrix mat;
point origin;
```

Six nab types—string, file, atom, residue, molecule and bounds use the predefined identifier NULL to indicate a non-existent object of these types. nab builtin functions returning objects of these types return NULL to indicate that the object could not be created. nab considers a NULL value to be false. The empty nab string "" is *not* equal to NULL.

36.2.2. Attributes

Four nab types—atom, residue, molecule and point—have attributes which are elements of their internal structure directly accessible at the nab level. Attributes are accessed via the select operator (.) which takes a variable as its left hand operand and an attribute name (an identifier) as its right. The general form is

```
var.attr
```

Most attributes behave exactly like ordinary variables of the same type. However, some attributes are read only. They are not permitted to appear as the left hand side of an assignment. When a read only attribute is passed to an nab function, it is copied into temporary variable which in turn is passed to the function. Read only attributes are not permitted to appear as destination variables in scanf() parameter lists. Attribute names are kept separate from variable and function names and since attributes can only appear to the right of select there is no conflict between variable and attribute names. For example, if x is a point, then

```
x // the point variable x
x.x // x coordinate of x
.x // Error!
```

Here is the complete list of nab attributes.

Atom attributes	Type	Write?	Meaning
atomname	string	yes	Ordinarily taken from columns 13-16 of an input pdb file, or from a residue library. Spaces are removed.
atomnum	int	no	The number of the atom starting at 1 for *each* strand in the molecule.
tatomnum	int	no	The *total* number of the atom starting at 1. Unlike atomnum, tatomnum does not restart at 1 for each strand.
fullname	string	no	The fully qualified atom name, having the form *strandnum*:*resnum*:*atomname*.
resid	string	yes	The *resid* of the residue containing this atom; see the **Residue attributes** table.
resname	string	yes	The name of the residue containing this atom.
resnum	int	no	The number of the residue containing the atom. resnum starts at 1 for *each* strand.
tresnum	int	no	The *total* number of the residue containing this atom starting at 1. Unlike resnum, tresnum does not restart at 1 for each strand.
strandname	string	yes	The name of the strand containing this atom.
strandnum	int	no	The number of the strand containing this atom.
pos	point	yes	point variable giving the atom's position.
x,y,z	float	yes	The Cartesian coordinates of this atom
charge	float	yes	Atomic charge
radius	float	yes	Dielectric radius
int1	int	yes	User-definable integer
float1	float	yes	User-definable float

Residue attributes	Type	Write?	Meaning
resid	string	yes	A 6-character string, ordinarily taken from columns 22-27 of a PDB file. It can be re-set to something else, but should always be either empty or exactly 6 characters long, since this string is used (if it is not empty) by *putpdb*.
resname	string	yes	Three-character identifier
resnum	int	no	The number of the residue. resnum starts at 1 for *each* strand.
tresnum	int	no	The *total* number of the residue, starting at 1. Unlike resnum, tresnum does not restart at 1 for each strand.
strandname	string	yes	The name of the strand containing this residue.
strandnum	int	no	The number of the strand containing this residue.

Molecule attributes	Type	Write?	Meaning
natoms	int	no	The total number of atoms in the molecule.
nresidues	int	no	The total number of residues in the molecule.
nstrands	int	no	The total number of strands in the molecule.

36.2.3. Arrays

nab supports two kinds of arrays—ordinary arrays where the selector is a comma separated list of integer expressions and associative or "hashed" arrays where the selector is a character string. The set of character strings that is associated with data in a hashed array is called its keys. Array elements may be of any nab type. All the dimensions of an ordinary array are indexed from 1 to *Nd* , where *Nd* is the size of the *d th* dimension. Non parameter array declarations are similar to scalar declarations except the variable name is followed by either a comma separated list of integer constants surrounded by square brackets ([]) for ordinary arrays or the reserved word hashed in square brackets for associative arrays. Associative arrays have no predefined size.

```
float energy[ 20 ], surface[ 13,13 ];
int attr[ dynamic, dynamic ];
molecule structs[ hashed ];
```

The syntax for multi-dimensional arrays like that for Fortran, not C. The *nab2c* compiler linearizes all index references, and the underlying C code sees only single-dimension arrays. Arrays are stored in "column-order", so that the most-rapidly varying index is the first index, as in Fortran. Multi-dimensional int or float arrays created in *nab* can generally be passed to Fortran routines expecting the analogous construct.

Dynamic arrays are not allocated space upon program startup, but are created and freed by the allocate and deallocate statements:

```
allocate attr[ i, j ];
....
deallocate attr;
```

Here i and j must be integer expressions that may be evaluated at run-time. It is an error (generally fatal) to refer to the contents of such an array before it has been allocated or after it has been deallocated.

36.2.4. Expressions

Expressions use operators to combine variables, constants and function values into new values. nab uses standard algebraic notation (a+b*c, etc) for expressions. Operators with higher precedence are evaluated first. Parentheses are used to alter the evaluation order. The complete list of nab operators with precedence levels and associativity is listed under **Operators**.

nab permits mixed mode arithmetic in that int and float data may be freely combined in expressions as long as the operation(s) are defined. The only exceptions are that the modulus operator (%) does not accept float operands, and that subscripts to ordinary arrays must be integer valued. In all other cases except parameter passing and assignment, when an int and float are combined by an operator, the int is converted to float then the operation is executed. In the case of parameter passing, nab requires (but does not check) that actual parameters passed to functions have the same type as the corresponding formal parameters. As for assignment (=) the right hand side is converted to the type of the left hand side (as long as both are numeric) and then assigned. nab treats assignment like any other binary operator which permits multiple assignments (a=b=c) as well as "embedded" assignments like:

```
if( mol = newmolecule() ) ...
```

nab relational operators are strictly binary. Any two objects can be compared provided that both are numeric, both are string or both are the same type. Comparisons for objects other than int, float and string are limited to tests for equality. Comparisons between file, atom, residue, molecule and bounds objects test for "pointer" equality, meaning that if the pointers are the same, the objects are same and thus equal, but if the pointers are different, no inference about the actual objects can be made. The most common comparison on objects of these types is against NULL to see if the object was correctly created. Note that as nab considers NULL to be false the following expressions are equivalent.

```
if( var == NULL )... is the same as if( !var )...
if( var != NULL )... is the same as if( var )...
```

The Boolean operators && and || evaluate only enough of an expression to determine its truth value. nab considers the value 0 to be false and *any* nonzero value to be true. nab supports direct assignment and concatenation of string values. The infix + is used for string concatenation.

nab provides several infix vector operations for point values. They can be assigned and point valued functions are permitted. Two point values can be added or subtracted. A point can be multiplied or divided by a float or an int. The unary minus can be applied to a point which has the same effect as multiplying it by -1. Finally, the at sign (@) is used to form the dot product of two points and the circumflex (^) is used to form their cross product.

36.2.5. Regular expressions

The =∼ and *!*∼ operators (match and not match) have strings on the left-hand-sides and *regular expression* strings on their right-hand-sides. These regular expressions are interpreted according to standard conventions drawn from the UNIX libraries.

36.2.6. Atom Expressions

An atom expression is a character string that contains one or more patterns that match a set of atom names in a molecule. Atom expressions contain three substrings separated by colons (:). They represent the strand, residue and atom parts of the atom expression. Each subexpression consists of a comma (,) separated list of patterns, or for the residue part, patterns and/or number ranges. Several atom expressions may be placed in a single character string by separating them with the vertical bar (|).

Patterns in atom expressions are similar to Unix shell expressions. Each pattern is a sequence of 1 or more single character patterns and/or stars (*). The star matches *zero* or more occurrences of *any* single character. Each part of an atom expression is composed of a comma separated list of limited regular expressions, or in the case of the residue part, limited regular expressions and/or ranges. A *range* is a number or a pair of numbers separated by a dash. A *regular expression* is a sequence of ordinary characters and "metacharacters". Ordinary characters represent themselves, while the metacharacters are operators used to construct more complicated patterns from the ordinary characters. All characters except ?, *, [,], -, ,(comma), : and | are ordinary characters. Regular expressions and the strings they match follow these rules.

aexpr	**matches**
x	An ordinary character matches itself.
?	A question mark matches any single character.
*	A star matches any run of zero of more characters. The pattern * matches anything.
[xyz]	A character class. It matches a single occurrence of any character between the [and the].
[^xyz]	A "negated" character class. It matches a single occurrence of any character not between the^ and the]. Character ranges, f-l , are permitted in both types of character class. This is a shorthand for all characters beginning with f up to and including l. Useful ranges are 0-9 for all the digits and a-zA-Z for all the letters.
-	The dash is used to delimit ranges in characters classes and to separate numbers in residue ranges.
$	The dollar sign is used in a residue range to represent the "last" residue without having to know its number.
,	The comma separates regular expressions and/or ranges in an atom expression part.
:	The colon separates the parts of an atom expression.
|	The vertical bar separates atom expressions in the same character string.
\	The backslash is used as an escape. Any character including metacharacters following a backslash matches itself.

Atom expressions match the *entire* name. The pattern C, matches only C, not CA, HC, etc. To match any name that begins with C use C*; to match any name that ends with C, use *C; to match any name containing a C, use *C*. A table of examples was given in chapter 2.

36.2.7. Format Expressions

A format expression is a special character string that is used to direct the conversion between the computer's internal data representations and their character equivalents. nab uses the underlying C compiler's printf()/scanf() system to provide formatted I/O. This section provides a short introduction to this system. For the complete description, consult any standard C reference. Note that since nab supports fewer types than its underlying C compiler, formatted I/O options pertaining to the data subtypes (h,l,L) are not applicable to nab format expressions.

An input format string is a mixture of ordinary characters, *spaces* and format descriptors. An output format string is mixture of ordinary characters including spaces and format descriptors. Each format descriptor begins with a percent sign (%) followed by several optional characters describing the format and ends with single character that specifies the type of the data to be converted. Here are the most common format descriptors. The ... represent optional characters described below.

%...c	convert a character
%...d	convert and integer
%...lf	convert a float
%...s	convert a string
%%	convert a literal %

Input and output format descriptors and format expressions resemble each other and in many cases the same format expression can be used for both input and output. However, the two types of format descriptors have different options and their actions are sufficiently distinct to consider in some detail. Generally, C based formatted output is more useful than C based formatted input.

When an input format expression is executed, it is scanned at most once from left to right. If the current format expression character is an ordinary character (anything but space or %), it must match the current character in the input stream. If they match then both the current character of the format expression and current character of the stream are advanced one character to the right. If they don't match, the scan ends. If the current format expression character is a space or a run of spaces and if the current input stream is one or more "white space" characters (space, tab, *newline*), then both the format and input stream are advanced to the next non-white space character. If the input format is one or more spaces but the current character of the input stream is non-blank, then only the format expression is advanced to the next non-blank character. If the current format character is a percent sign, the format descriptor is used to convert the next "field" in the input stream. A field is a sequence of non-blank characters surrounded by white space or the beginning or end of the stream. This means that a format descriptor will *skip* white space including newlines to find non blank characters to convert, even if it is the first element of the format expression. This implicit scanning is what limits the ability of C based formatted input to read fixed format data that contains any spaces.

Note that lf is used to input a NAB *float* variable, rather than the f argument that would be used in C. This is because *float* in NAB is converted to *double* in the output C code (see *defreal.h* if you want to change this behavior.) Ideally, the NAB compiler should parse the format string, and make the appropriate substitutions, but this is not (yet) done: NAB translates the format string directly into the C code, so that the NAB code must also generally use lf as a format descriptor for floating point values.

nab input format descriptors have two options, a field width, and an assignment suppression indicator. The field width is an integer which specifies how much of current *field* and not the input stream is to be converted. Conversion begins with the first character of the field and stops when the correct number of characters have been converted or white space is encountered. A star (*) option indicates that the field is to be converted, but the result of the conversion is not stored. This can be used to skip unwanted items in a data stream. The order of the two options does not matter.

The execution of an output format expression is somewhat different. It is scanned once from left to right. If the current character is not a percent sign, it placed on the output stream. Thus spaces have no special significance in formatted output. When the scan encounters a percent sign it replaces the entire format descriptor with the properly formatted value of the corresponding output expression.

Each output format descriptor has four optional attributes—width, alignment, padding and precision. The width is the *minimum* number of characters the data is to occupy for output. Padding controls how the field will be filled if the number of characters required for the data is less than the field width. Alignment specifies whether the data is to start in the first character of the field (left aligned) or end in the last (right aligned). Finally precision, which applies only to string and float conversions controls how much of the string is be converted or how many digits should follow the decimal point.

Output field attributes are specified by optional characters between the initial percent sign and the final data type character. Alignment is first, with left alignment specified by a minus sign (-). Any other character after the percent sign indicates right alignment. Padding is specified next. Padding depends on both the alignment and the type of the data being converted. Character conversions (`%c`) are always filled with spaces, regardless of their alignment. Left aligned conversions are also always filled with spaces. However, right aligned string and numeric conversions can use a `0` to indicate that left fill should be zeroes instead of spaces. In addition numeric conversions can also specify an optional + to indicate that non-negative numbers should be preceded by a plus sign. The default action for numeric conversions is that negative numbers are preceded by a minus, and other numbers have no sign. If both `0` and + are specified, their order does not matter.

Output field width and precision are last and are specified by one or two integers or stars (*) separated by a period (.). The first number (or star) is the field width, the second is its precision. If the precision is not specified, a default precision is chosen based on the conversion type. For floats (%f), it is six decimal places and for strings it is the entire string. Precision is not applicable to character or integer conversions and is ignored if specified. Precision may be specified without the field width by use of single integer (or star) preceded by a period. Again, the action is conversion type dependent. For strings (`%s`), the action is to print the first *N* characters of the string or the entire string, whichever is shorter. For floats (%f), it will print *N* decimal places but will extend the field to whatever size if required to print the whole number part of the float. The use of the star (*) as an output width or precision indicates that the width or precision is specified as the next argument in the conversion list which allows for runtime widths and precisions.

Ouput format options	
Alignment	
-	left justified
default	right justified
Padding	
0	%d, %f, `%s` only, left fill with zeros, right fill with spaces.
+	`%d`, %f only, precede non-negative numbers with a +.
default	left and right fill with spaces.
Width & precision	
W	*minimum* field width of W. W is either an integer or a * where the star indicates that the width is the next argument in the parameter list.
W.P	*minimum* field width of W, with a precision of P. W,P are integers or stars, where stars indicate that they are to be set from the appropriate arguments in the parameter list. Precision is ignored for `%c` and `%d`.
.P	`%s`, print the first *P* characters of the string or the entire string whichever is shorter. %f, print *P* decimal places in a field wide enough to hold the integer and fractional parts of the number. `%c` and `%d`, use whatever width is required. Again *P* is either an integer or a star where the star indicates that it is to be taken from the next expression in the parameter list.
default	`%c`, `%d`, `%s`, use whatever width is required to exactly hold the data. %f, use a precision of 6 and whatever width is required to hold the data.

36.3. Statements

nab statements describe the action the nab program is to perform. The expression statement evaluates expressions. The if statement provides a two way branch. The while and for statements provide loops. The break statement is used to “short circuit” or exit these loops. The continue statement advances a for loop to its next iteration. The return statement assigns a function’s value and returns control to the caller. Finally a list of statements can be enclosed in braces ({}) to create a compound statement.

36.3.1. Expression Statement

An expression statement is an expression followed by a semicolon. It evaluates the expression. Many expression statements include an assignment operator and its evaluation will update the values of those variables on the left hand side of the assignment operator. These kinds of expression statements are usually called “assignment statements” in other languages. Other expression statements consist of a single function call with its result ignored. These statements take the place of “call statements” in other languages. Note that an expression statement can contain *any* expression, even ones that have no lasting effect.

```
mref = getpdb( "5p21.pdb" ); // "assignment" stmt
m = getpdb( "6q21.pdb" );
superimpose( m,"::CA",mref,"::CA" ); // "call" stmt
0; // expression stmt.
```

36.3.2. Delete Statement

nab provides the delete statement to remove elements of hashed arrays. The syntax is

```
delete h_array[ str ];
```

where *h_array* is a hashed array and *str* is a string valued expression. If the specified element is in *h_array* it is removed; if not, the statement has no effect.

36.3.3. If Statement

The if statement is used to choose between two options based on the value of the if expression. There are two kinds of if statements—the simple if and the if-else. The simple if contains an expression and a statement. If the expression is true (any nonzero value), the statement is executed. If the expression is false (0), the statement is skipped.

```
if( expr ) true_stmt;
```

The if-else statement places two statements under control of the if. One is executed if the expression is true, the other if it is false.

```
if( expr )
   true_stmt;
else
   false_stmt;
```

36.3.4. While Statement

The while statement is used to execute the statement under its control as long as the the while expression is true (nonzero). A compound statement is required to place more than one statement under the while statement’s control.

```
while( expr ) stmt;
while( expr ) {
   stmt_1;
   stmt_2;
   ...
   stmt_N;
}
```

36.3.5. For Statement

The for statement is a loop statement that allows the user to include initialization and an increment as well as a loop condition in the loop header. The single statement under the control of the for statement is executed as long as the condition is true (nonzero). A compound statement is required to place more than one statement under control of a for. The general form of the for statement is

```
for( expr_1; expr_2; expr_3 )  stmt;
```

which behaves like

```
expr_1;
while( expr_2 ) {
     stmt;
     expr_3;
}
```

expr_3 is generally an expression that computes the next value of the loop index. Any or all of *expr_1*, *expr_2* or *expr_3* can be omitted. An omitted *expr_2* is considered to be true, thus giving rise to an “infinite” loop. Here are some for loops.

```
for( i = 1; i <= 10; i = i + 1 )
printf( "%3d\n", i ); // print 1 to 10
for( ; ; ) // "infinite" loop
{
   getcmd( cmd ); // Exit better be in
   docmd( cmd ); // getcmd() or docmd().
}
```

nab also includes a special kind of for statement that is used to range over all the entries of a hashed array or all the atoms of a molecule. The forms are

```
// hashed version
for( str in h_array ) ~stmt;
// molecule version
for( a in mol ) ~stmt;
```

In the first code fragment, *str* is string and *h_array* is a hashed array. This loop sets *str* to each key or string associated with data in *h_array*. Keys are returned in increasing lexical order. In the second code fragment *a* is an atom and *mol* is a molecule. This loop sets *a* to each atom in *mol*. The first atom is the first atom in the first residue of the first strand. Once all the atoms in this residue have been visited, it moves to the first atom of the next residue in the first strand. Once all atoms in all residues in the first strand have been visited, the process is repeated on the second and subsequent strands in mol until all atoms have been visited. The order of the strands of molecule is the order in which they were created using addstrand(). Residues in each strand are numbered from 1 to *N*. The order of the atoms in a residue is the order in which the atoms were listed in the reslib entry or pdbfile that that residue derives from.

36.3.6. Break Statement

Execution of a break statement exits the immediately enclosing for or while loop. By placing the break under control of an if conditional exits can be created. break statements are only permitted inside while or for loops.

```
for( expr_1; expr_2; expr_3 ) {
    ...
    if( expr ) break; // "break" out of loop
      ...
}
```

36.3.7. Continue Statement

Execution of a continue statement causes the immediately enclosing for loop to skip to its next value. If the next value causes the loop control expression to be false, the loop is exited. continue statements are permitted only inside while and for loops.

```
for( expr_1; expr_2; expr_3 ) {
    ... if( expr ) continue; // "continue" with next value
    ...
}
```

36.3.8. Return Statement

The return statement has two uses. It terminates execution of the current function returning control to the point immediately following the call and when followed by an optional expression, returns the value of the expression as the value of the function. A function's execution also ends when it “runs off the bottom”. When a function executes the last statement of its definition, it returns even if that statement is not a return. The value of the function in such cases is undefined.

```
return expr; // return the value expr
return; // return, function value undefined.
```

36.3.9. Compound Statement

A compound statement is a list of statements enclosed in braces. Compound statements are required when a loop or an if has to control more than one statement. They are also required to associate an else with an if other than the nearest unpaired one. Compound statements may include other compound statements. Unlike C, nab compound statements are not blocks and may not include declarations.

36.4. Structures

A struct is collection of data elements, where the elements are accessed via their names. Unlike arrays which require all elements of an array to have the same type, elements of a structure can have different types. Users define a struct via the reserved word ‘struct’. Here's a simple example, a struct that could be used to hold a complex number.

```
struct cmplx_t { float r, i; } c;
```

This declares a nab variable, ‘c’, of user defined type ‘struct cmplx_t’. The variable, c, has two float valued elements, ‘c.r’, ‘c.i’ which can be used like any other nab float variables:

```
c.r = -2.0; ... 5*c.i ... printf( "c.r,i = %8.3f, %8.3f\n", c.r, c.i );
```

Now, let's look more closely at that struct declaration.

```
struct cmplx_t { float r, i; } c;
```

As mentioned before, every nab struct begins with the reserved word struct. This must be followed by an identifier called the structure tag, which in this example is ‘cmplx_t’. Unlike C/C++, a nab struct can not be anonymous.

Following the structure tag is a list of the struct’s element declarations surrounded by a left and right curly bracket. Element declarations are just like ordinary nab variable declarations: they begin with the type, followed by a comma separated list of variables and end with a semicolon. nab structures must contain at least one declaration containing at least one variable. Also, nab struct elements are currently restricted to scalar values of the basic nab types, so nab structs can not contain arrays or other structs. Note that in our example, both elements are in one declaration, but two declarations would have worked as well.

The whole assembly ‘struct ... }’ serves to define a new type which can be used like any other nab type to declare variables of that type, in this example, a single scalar variable, ‘c’. And finally, like all other nab variable declarations, this one also ends with a semicolon.

Although nab structs can not contain arrays, nab allows users to create arrays, including dynamic and hashed arrays of structs. For example

```
struct cmplx_t { float r, i; } a[ 10 ], da[ dynamic ], ha[ hashed ];
```

declares an ordinary, dynamic and hashed array of struct cmplx_t.

Up til now, we’ve only looked at complete struct declaration. Our example

```
struct cmplx_t { float r, i; } c;
```

contains all the parts of a struct declaration. However there are two other forms of struct declarations. The first one is to define a type, as opposed to declaring variables:

```
struct cmplx_t { float r, i; };
```

defines a new type ‘struct cmplx_t’ but does not declare any variables of this type. This is quite useful in that the type can be placed in a header file allowing it to be shared among parts of a larger program.

The othe form of a struct declaration is this short form:

```
struct cmplx_t cv1, cv2;
```

This form can only be used once the type has been defined, either via a type declaration (ie not variable) or a complete type + variable declaration. In fact, once a struct type has been defined, all subsequent declarations of variables of that type, including parameters, must use the short form.

```
struct cmplx_t { float r, i; }; // define type type `struct cmplx_t'
struct cmplx_t c, ctab[ 10 ]; // define some vars
int f( int s, struct cmplx_t ct[1] ) // func taking array of
                                     // struct cmplx_t { ... };
```

36.5. Functions

A function is a named group of declarations and statements that is executed as a unit by using the function’s name in an expression. Functions may include special variables called parameters that enable the same function to work on different data. All nab functions return a value which can be ignored in the calling expression. Expression statements consisting of a single function call where the return value is ignored resemble procedure call statements in other languages.

All parameters to user defined nab functions are passed by reference. This means that each nab parameter operates on the actual data that was passed to the function during the call. Changes made to parameters during the execution of the function will persist after the function returns. The only exception to this is if an expression is passed in as a parameter to a user defined nab function. It this case, nab evaluates the expression, stores its value in a compiler created temporary variable and uses that temporary variable as the actual parameter. For example if a user were to pass in the constant 1 to an nab function which in turned used it and then assigned it the value 6, the 6 would be stored in the temporary location and the external 1 would be unchanged.

36.5.1. Function Definitions

An nab function definition begins with a header that describes the function value type, the function name and the parameters if any. If a function does not have parameters, an empty parameter list is still required. Following the header is a list of declarations and statements enclosed in braces. The function's declarations must precede all of its statements. A function can include zero or more declarations and/or zero or more statements. The empty function—no declarations and no statements is legal.

The function header begins with the reserved word specifying the type of the function. All nab functions must be typed. An nab function can return a single value of any nab type. nab functions can not return nab arrays. Following the type is an identifier which is the name of the function. Each parameter declaration begins with the parameter type followed by its name. Parameter declarations are enclosed in parentheses and separated by commas. If a function has no parameters, there is nothing between the parentheses. Here is the general form of a function definition:

```
ftype fname( ptype1 parm1, ... )
{
   decls
   stmts
};
```

36.5.2. Function Declarations

nab requires that every function be declared or made known to the compiler before it is used. Unfortunately this is not possible if functions used in one source file are defined in other source files or if two functions are mutually recursive. To solve these problem, nab permits functions to be declared as well as defined. A function declaration resembles the header of a function definition. However, in place of the function body, the declaration ends with a semicolon or a semicolon preceded by either the word c or the word fortran indicating the external function is written in C or Fortran instead of nab.

```
ftype fname( ptype1 parm1, ... ) flang;
```

36.6. Points and Vectors

The nab type point is an object that holds three float values. These values can represent the X, Y and Z coordinates of a point or the components of 3-vector. The individual elements of a point variable are accessed via attributes or suffixes added to the variable name. The three point attributes are "x", "y" and "z". Many nab builtin functions use, return or create point values. When used in this context, the three attributes represent the point's X, Y and Z coordinates. nab allows users to combine point values with numbers in expressions using conventional algebraic or infix notation. nab does not support operations between numbers and points where the number must be converted into a vector to perform the operation. For example, if p is a point then the expression p + 1. is an error, as nab does not know how to expand the scalar 1. into a 3-vector. The following table contains nab point and vector operations. p, q are point variables; *s* a numeric expression.

Operator	Example	Precedence	Explanation
Unary -	-p	8	Vector negation, same as -1 * p.
^	p^ q	7	Compute the cross or vector product of p, q.
@	p @ q	6	Compute the scalar or dot product of p, q.
*	*s* * p	6	Multiply p by *s*, same as p * *s*.
/	p / *s*	6	Divide p by *s*, s / p not allowed.
+	p + q	5	Vector addition
Binary -	p - q	5	Vector subtraction
==	p == q	4	Test if p and q equal.
!=	p != q	4	Test if p and q are different.
=	p = q	1	Set the value of p to q.

36.7. String Functions

nab provides the following awk-like string functions. Unlike awk, the nab functions do not have optional parameters or builtin variables that control the actions or receive results from these functions. nab strings are indexed from 1 to *N* where *N* is the number of characters in the string.

int length(string s);
int index(string s, string t);
int match(string s, string r, int rlength);
string substr(string s, int pos, int len);
int split(string s, string fields[], string fsep);
int sub(string r, string s, string t);
int gsub(string r, string s, string t);

length() returns the length of the string s. Both "" and NULL have length 0. index() returns the position of the left most occurrence of t in s. If t is not in s, index() returns 0. match returns the position of the longest leftmost substring of s that matches the regular expression r. The length of this substring is returned in rlength. If no substring of s matches r, match() returns 0 and rlength is set to 0. substr() extracts the substring of length len from s beginning at position pos. If len is greater than the rest of the string beginning at pos, return the substring from pos to *N* where *N* is the length of the string. If pos is < 1 or $> N$, return "".

split() partitions s into fields separated by fsep. These field strings are returned in the array fields. The number of fields is returned as the function value. The array fields must be allocated before split() is called and must be large enough to hold all the field strings. The action of split() depends on the value of fsep. If fsep is a string containing one or more blanks, the fields of s are considered to be separated by *runs* of white space. Also, leading and trailing white space in s do not indicate an empty initial or final field. However, if fsep contains any value but blank, then fields are considered to be delimited by *single* characters from fsep and initial and/or trailing fsep characters do represent initial and/or trailing fields with values of "". NULL and the empty string "" have 0 fields. If both s and fsep are composed of only white space then s also has 0 fields. If fsep is not white space and s consists of nothing but characters from fsep, s will have $N + 1$ fields of "" where *N* is the number of characters of s.

sub() replaces the leftmost, longest substring of the *target string* t that matches the *regular expression* r with the *substitution string* s. gsub() replaces all non-overlapping substrings of t that match the regular expression r with the string s. Each function returns the number of substitutions made. Unlike awk, the regular expression r is a string variable, with no surrounding '/' characters. For example:

```
int nmatch;
string regexp, substitute, target;
target = “water, water, everywhere”;
regexp = “at”;
substitute = “ith”;
nmatch = gsub( regexp, substitute, target);
```

After this, `target` will contain “`wither, wither, everywhere`”, and `nmatch` will be 2.

The special substitute character '&' stands for the precise substring matched by the regular expression. Hence

```
target = “daabaaa”;
sub( “a+”, “c&c”, target);
```

will yield “`dcaacbaaa`”. Note what “leftmost, longest substring” means here: “lefmost” takes precedence over “longest”.

36.8. Math Functions

nab provides the builtin mathematical functions shown in Table 36.1. Since nab is intended for chemical structure calculations which always measure angles in degrees, the argument to the trig functions—cos(), sin() and tan()— and the return value of the inverse trig functions—acos(), asin(), atan() and atan2()—are in degrees instead of radians as they are in other languages. Note that the pseudo-random number functions have a different calling sequence than in earlier versions of NAB; you may have to edit and re-compile earlier programs that used those routines.

36.9. System Functions

int exit(int i);
int system(string cmd);

The function exit() terminates the calling nab program with return status i. system() invokes a subshell to execute cmd. The subshell is always /bin/sh. The return value of system() is the return value of the subshell and not the command it executed.

36.10. I/O Functions

nab uses the C I/O model. Instead of special I/O statements, nab I/O is done via calls to special builtin functions. These function calls have the same syntax as ordinary function calls but some of them have different semantics, in that they accept both a variable number of parameters and the parameters can be various types. nab uses the underlying C compiler's printf()/scanf() system to perform I/O on int, float and string objects. I/O on point is via their float x, y and z attributes. molecule I/O is covered in the next section, while bounds can be written using dumpbounds(). Transformation matrices can be written using dumpmatrix(), but there is currently no builtin for reading them. The value of an nab file object may be written by treating as an integer. Input to file variables is not defined.

36.10.1. Ordinary I/O Functions

nab provides these functions for stream or FILE * I/O of int, float and string objects.

int fclose(file f);
file fopen(string fname, string mode);
int unlink(string fname);
int printf(string fmt, ...);
int fprintf(file f, string fmt, ...);
string sprintf(string fmt, ...);
int scanf(string fmt, ...);
int fscanf(file f, string fmt, ...);
int sscanf(string str, string fmt, ...);
string getline(file f);

fclose() closes (disconnects) the file represented by f. It returns 0 on success and -1 on failure. All open nab files are automatically closed when the program terminates. However, since the number of open files is limited, it is a good idea to close open files when they are no longer needed. The system call unlink removes (deletes) the file.

fopen() attempts to open (prepare for use) the file named fname with mode mode. It returns a valid nab file on success, and NULL on failure. Code should thus check for a return value of NULL, and do the appropriate thing. (An alternative, safe_fopen() sends an error message to stderr and exits on failure; this is sometimes a convenient alternative to fopen() itself, fitting with a general bias of nab system functions to exit on failure, rather than to return error codes that must always be processed.) Here are the most common values for mode and their meanings. For other values, consult any standard C reference.

Inverse Trig Functions.	
float acos(float x);	Return $\cos^{-1}(x)$ in degrees.
float asin(float x);	Return $\sin^{-1}(x)$ in degrees.
float atan(float x);	Return $\tan^{-1}(x)$ in degrees.
float atan2(float x);	Return tan -1 (y / x) in degrees. By keeping x and y separate, 90o can be returned without encountering a zero divide. Also, atan2 will return an angle in the full range [-180o, 180o].
Trig Functions	
float cos(float x);	Return cos(x), where x is in degrees.
float sin(float x);	Return sin(x), where x is in degrees.
float tan(float x);	Return tan(x), where x is in degrees.
Conversion Functions.	
float atof(string str);	Interpret the next run of non blank characters in str as a float and return its value. Return 0 on error.
int atoi(string str);	Interpret the next run of non blank characters in str as an int and return its value. Return 0 on error.
Other Functions.	
float rand2();	Return a pseudo-random number in (0,1).
float gauss(float mean, float sd);	Return a pseudo-random number taken from a Gaussian distribution with the given mean and standard deviation. The rand2() and gauss() routines share a common seed.
int setseed(int seed);	Reset the pseudo-random number sequence with the new seed, which must be a negative integer.
int rseed( );	Use the system time() command to set the random number sequence with a reasonably random seed. Returns the seed it used; this could be used in a later call to setseed() to regenerate the same sequence of pseudo-random values.
float ceil(float x);	Return $\lceil x \rceil$.
float exp(float x);	Return e^x.
float cosh(float x);	Return the hyperbolic cosine of x.
float fabs(float x);	Return $\|x\|$.
float floor(float x);	Return $\lfloor x \rfloor$.
float fmod(float x, float y);	Return r, the remainder of x with respect to y; the signs of r and y are the same.
float log(float x);	Return the natural logarithm of x.
float log10(float x);	Return the base 10 logarithm of x.
float pow(float x, float y);	Return x^y, $x > 0$.
float sinh(float x);	Return the hyperbolic sine of x.
float tanh(float x);	Return the hyperbolic tangent of x.
float sqrt(float x);	Return positive square root of x, x >= 0.

Table 36.1.: *NAB built-in mathematical functions*

fopen() mode values	
“r”	Open for reading. The file fname must exist and be readable by the user.
“w”	Open for writing. If the file exists and is writable by the user, truncate it to zero length. If the file does not exist, and if the directory in which it will exist is writable by the user, then create it.
“a”	Open for appending. The file must exist and be writable by the user.

The three functions printf(), fprintf() and sprintf() are for formatted (ASCII) output to stdout, the file f and a string. Strictly speaking, sprintf() does not perform output, but is discussed here because it acts as if “writes” to a string. Each of these functions uses the format string fmt to direct the conversion of the expressions that follow it in the parameter list. Format strings and expressions are discussed **Format Expressions**. The first format descriptor of fmt is used to convert the first expression after fmt, the second descriptor, the next expression etc. If there are more expressions than format descriptors, the extra expressions are not converted. If there are fewer expressions than format descriptors, the program will likely die when the function tries to covert non-existent data.

The three functions scanf(), fscanf() and sscanf() are for formatted (ASCII) input from stdin, the file f and the string str. Again, sscanf() does not perform input but the function behaves like it is “reading” from str. The action of these functions is similar to their output counterparts in that the format expression in fmt is used to direct the conversion of characters in the input and store the results in the variables specified by the parameters following fmt. Format descriptors in fmt correspond to variables following fmt, with the first descriptor corresponding to the first variable, etc. If there are fewer descriptors than variables, then extra variables are not assigned; if there are more descriptors than variables, the program will most likely die due to a reference to a non-existent address.

There are two very important differences between nab formatted I/O and C formatted I/O. In C, formatted input is assigned through pointers to the variables (&*var*). In nab formatted I/O, the compiler automatically supplies the addresses of the variables to be assigned The second difference is when a string object receives data during an nab formatted I/O. nab strings are allocated when needed. However, in the case of any kind of scanf() to a string or the implied (and hidden) writing to a string with sprintf(), the number of characters to be written to the string is unknown until the string has been written. nab automatically allocates strings of length 256 to hold such data with the idea that 256 is usually big enough. However, there will be cases where it is not big enough and this will cause the program to die or behave strangely as it will overwrite other data.

Also note that the default precision for floats in *nab* is double precision (see $AMBERHOME/AmberTools/src/nab/defreal.h, since this could be changed, or may be different on your system.) Formats for floats for the *scanf* functions then need to be "%lf" rather than "%f".

The getline() function returns a string that has the next line from file f. The end-of-line character has been stripped off.

36.10.2. matrix I/O

NAB uses 4x4 matrices to represent coordinate transformations:

```
  r  r  r 0
  r  r  r 0
  r  r  r 0
 dx dy dz 1
```

The r’s are a 3x3 rotation matrix, and the d’s are the translations along the X,Y and Z axes.

NAB coordinates are row vectors which are transformed by appending a 1 to each point (x,y,z) -> (x,y,z,1), post multiplying by the transformation matrix, and then discarding the final 1 in the new point.

Two builtins are provided for reading/writing transformation matrices.

```
matrix getmatrix(string filename);
```

Read the matrix from the file with name filename. Use "-" to read a matrix from stdin. A matrix is 4 lines of 4 numbers. A line of less than 4 numbers is an error, but anything after the 4th number is ignored. Lines beginning with a '#' are comments. Lines after the 4th data line are not read. Return a matrix with all zeroes on error, which can be tested:

```
mat = getmatrix("bad.mat");
if(!mat){ fprintf(stderr, "error reading matrix\n"); ... }
```

Keep in mind that nab transformations are intended for use on molecular coordinates, and that transformations like scaling and shearing [which can not be created with nab directly but can now be introduced via *getmatrix*()] may lead to incorrect on non-sensical results.

```
int putmatrix(string filename, matrix mat);
```

Write matrix mat to to file with name filename. Use "-" to write a matrix to stdout. There is currently no way to write matrix to stderr. A matrix is writen as 4 lines of 4 numbers. Return 0 on success and 1 on failure.

36.11. Molecule Creation Functions

The nab molecule type has a complex and dynamic internal structure organized in a three level hierarchy. A molecule contains zero or more named strands. Strand names are strings of any characters except white space and can not exceed 255 characters in length. Each strand in a molecule must have a unique name. Strands in different molecules may have the same name. A strand contains zero or more residues. Residues in each strand are numbered from 1. There is no upper limit on the number of residues a strand may contain. Residues have names, which need not be unique. However, the combination of *strand-name*:*res-num* is unique for every residue in a molecule. Finally residues contain one or more atoms. Each atom name in a residue should be distinct, although this is neither required nor checked by nab. nab uses the following functions to create and modify molecules.

molecule newmolecule();
molecule copymolecule(molecule mol);
int freemolecule(molecule mol);
int freeresidue(residue r);
int addstrand(molecule mol, string sname);
int addresidue(molecule mol, string sname, residue res);
int connectres(molecule mol, string sname, int res1, string aname1,
int res2, string aname2);
int mergestr(molecule mol1, string str1, string end1, molecule mol2, string str2, string end2);

newmolecule() creates an "empty" molecule—one with no strands, residues or atoms. It returns NULL if it can not create it. copymolecule() makes a copy of an existing molecule and returns a NULL on failure. freemolecule() and *freeresidue()* are used to deallocate memory set aside for a molecule or residue. In most programs, these functions are usually not necessary, but should be used when a large number of molecules are being copied. Once a molecule has been created, addstrand() is used to add one or more named strands. Strands can be added at any to a molecule. There is no limit on the number of strands in a molecule. Strands can be added to molecules created by getpdb() or other functions as long as the strand names are unique. addstrand() returns 0 on success and 1 on failure. Finally addresidue() is used to add residues to a strand. The first residue is numbered 1 and subsequent residues are numbered 2, 3, etc. addresidue() also returns 0 on success and 1 on failure.

nab requires that users explicitly make all inter-residue bonds. connectres() makes a bond between two atoms of *different* residues of the strand with name sname. It returns 0 on success and 1 on failure. Atoms in different strands can not be bonded. The bonding between atoms in a residue is set by the residue library entry and can not be changed at runtime at the nab level.

The last function mergestr() is used to merge two strands of the same molecule or copy a strand of the second molecule into a strand of the first. The residues of a strand are ordered from 1 to *N*, where *N* is the number of residues in that strand. nab imposes no chemical ordering on the residues in a strand. However, since the strands are generally ordered, there are four ways to combine the two strands. mergestr() uses the two values "first" and "last" to stand for residues 1 and *N*. The four combinations and their meanings are shown in the next table. In the table, str1 has *N* residues and str2 has *M* residues.

end1	**end2**	**Action**
first	first	The residues of str2 are reversed and then inserted before those of str1: M , ..., 2, 1 : 1 , 2 , ..., N
first	last	The residues of str2 are inserted before those of str1: 1 , 2, ..., M : 1 , 2 , ..., N
last	first	The residues of str2 are inserted after those of str1: 1 , 2 , ..., N : 1 , 2 , ..., M
last	last	The residues of str2 are reversed and then inserted after those of str1: 1 , 2 , ..., N : M , ..., 2 , 1

36.12. Creating Biopoloymers

molecule linkprot(string strandname, string seq, string reslib);
molecule link_na(string strandname, string seq, string reslib, string natype, string opts);

Although many nab functions don't care what kind of molecule they operate on, many operations require molecules that are compatible with the Amber force field libraries (see Chapter 6). The best and most general way to do this is to use tleap commands, described in Chapter 8). The *linkprot()* and *link_na()* routines given here are limited commands that may sometimes be useful, and are included for backwards compatibility with earlier versions of NAB.

linkprot() takes a strand identifier and a sequence, and returns a molecule with this sequence. The molecule has an extended structure, so that the ϕ, $\psi$and ω angles are all 180o . The *reslib* input determines which residue library is used; if it is an empty string, the AMBER 94 all-atom library is used, with charged end groups at the N and C termini. All nab residue libraries are denoted by the suffix .rlb and LEaP residue libraries are denoted by the suffix .lib. If *reslib* is set to "nneut", "cneut" or "neut", then neutral groups will be used at the N-terminus, the C-terminus, or both, respectively.

The seq string should give the amino acids using the one-letter code with upper-case letters. Some non-standard names are: "H" for histidine with the proton on the δ position; "h" for histidine with the proton at the ε position; "3" for protonated histidine; "n" for an acetyl blocking group; "c" for an HNMe blocking group, "a" for an NH 2 group, and "w" for a water molecule. If the sequence contains one or more "|" characters, the molecule will consist of separate polypeptide strands broken at these positions.

The link_na() routine works much the same way for DNA and RNA, using an input residue library to build a single-strand with correct local geometry but arbitrary torsion angles connecting one residue to the next. natype is used to specify either DNA or RNA. If the *opts* string contains a "5", the 5' residue will be "capped" (a hydrogen will be attached to the O5' atom); if this string contains a "3" the O3' atom will be capped.

The newer (and generally recommended) way to generate biomolecules uses the *getpdb_prm()* function described in Chapter 39.

36.13. Fiber Diffraction Duplexes in NAB

The primary function in NAB for creating Watson-Crick duplexes based on fibre-diffraction data is fd_helix:

molecule fd_helix(string helix_type, string seq, string acid_type);

fd_helix() takes as its arguments three strings - the helix type of the duplex, the sequence of one strand of the duplex, and the acid type (which is "dna" or "rna"). Available helix types are as follows:

Helix type options for fd_helix()	
arna	Right Handed A-RNA (Arnott)
aprna	Right Handed A'-RNA (Arnott)
lbdna	Right Handed B-DNA (Langridge)
abdna	Right Handed B-DNA (Arnott)
sbdna	Left Handed B-DNA (Sasisekharan)
adna	Right Handed A-DNA (Arnott)

The molecule returns contains a Watson-Crick double-stranded helix, with the helix axis along z. For a further explanation of the fd_helix code, please see the code comments in the source file fd_helix.nab.

References for the fibre-diffraction data:

1. Structures of synthetic polynucleotides in the A-RNA and A'-RNA conformations. X-ray diffraction analyses of the molecule conformations of (polyadenylic acid) and (polyinosinic acid).(polycytidylic acid). Arnott, S.; Hukins, D.W.L.; Dover, S.D.; Fuller, W.; Hodgson, A.R. *J.Mol. Biol.* (1973), 81(2), 107-22.

2. Left-handed DNA helices. Arnott, S; Chandrasekaran, R; Birdsall, D.L.; Leslie, A.G.W.; Ratliff, R.L. *Nature* (1980), 283(5749), 743-5.

3. Stereochemistry of nucleic acids and polynucleotides. Lakshimanarayanan, A.V.; Sasisekharan, V. *Biochim. Biophys. Acta* 204, 49-53.

4. Fuller, W., Wilkins, M.H.F., Wilson, H.R., Hamilton, L.D. and Arnott, S. (1965). *J. Mol. Biol.* 12, 60.

5. Arnott, S.; Campbell Smith, P.J.; Chandraseharan, R. in *Handbook of Biochemistry and Molecular Biology, 3rd Edition. Nucleic Acids–Volume II*, Fasman, G.P., ed. (Cleveland: CRC Press, 1976), pp. 411-422.

36.14. Reduced Representation DNA Modeling Functions

nab provides several functions for creating the reduced representation models of DNA described by R. Tan and S. Harvey.[651] This model uses only 3 pseudo-atoms to represent a base pair. The pseudo atom named CE represents the helix axis, the atom named SI represents the position of the sugar-phosphate backbone on the sense strand and the atom named MA points into the major groove. The plane described by these three atoms (and a corresponding virtual atom that represents the anti sugar-phosphate backbone) represents quite nicely an all atom watson-crick base pair plane.

molecule dna3(int nbases, float roll, float tilt, float twist, float rise);
molecule dna3_to_allatom(molecule m_dna3, string seq, string aseq, string reslib, string natype);
molecule allatom_to_dna3(molecule m_allatom, string sense, string anti);

The function dna3() creates a reduced representation DNA structure. dna3() takes as parameters the number of bases nbases, and four helical parameters roll, tilt, twist, and rise.

dna3_to_allatom() makes an all-atom dna model from a dna3 molecule as input. The molecule m_dna3 is a dna3 molecule, and the strings seq and aseq are the sense and anti sequences of the all-atom helix to be constructed. Obviously, the number of bases in the all-atom model should be the same as in the dna3 model. If the string aseq is left blank (""), the sequence generated is the wc_complement() of the sense sequence. reslib names the residue library from which the all-atom model is to be constructed. If left blank, this will default to all_nucleic94.lib The last parameter is either "dna" or "rna" and defaults to dna if left blank.

The allatom_to_dna3() function creates a dna3 model from a double stranded all-atom helix. The function takes as parameters the input all-atom molecule m_allatom, the name of the sense strand in the all-atom molecule, sense and the name of the anti strand, anti.

36.15. Molecule I/O Functions

nab provides several functions for reading and writing molecule and residue objects.

residue getresidue(string rname, string rlib);
molecule getpdb(string fname [, string options]);
molecule getcif(string fname, string blockId);
int putpdb(string fname, molecule mol [, string options]);
int putcif(string fname, molecule mol);
int putbnd(string fname, molecule mol);
int putdist(string fname, molecule mol);

The function getresidue() returns a copy of the residue with name rname from the residue library named rlib. If it can not do so it returns the value NULL.

The function getpdb() converts the contents of the PDB file with name fname into an nab molecule. getpdb() creates bonds between any two atoms in the same residue if their distance is less than: 1.20 Å if either atom is a hydrogen, 2.20 Å if either atom is a sulfur, and 1.85 Å otherwise. Atoms in different residues are never bonded by getpdb().

getpdb() creates a new strand each time the chain id changes or if the chain id remains the same and a TER card is encountered. The strand name is the chain id if it is not blank and "*N*", where *N* is the number of that strand in the molecule beginning with 1. For example, a PDB file containing chain with no chain ID, followed by chain A, followed by another blank chain would have three strands with names "1", "A" and "3". getpdb() returns a molecule on success and NULL on failure.

The optional final argument to getpdb can be used for a variety of purposes, which are outlined in the table below.

The (experimental!) function getcif is like getpdb, but reads an mmCIF (macro-molecular crystallographic information file) formatted file, and extracts "atom-site" information from data block blockID. You will need to compile and install the cifparse library in order to use this.

The next group of builtins write various parts of the molecule mol to the file fname. All return 0 on success and 1 on failure. If fname exists and is writable, it is overwritten without warning. putpdb() writes the molecule mol into the PDB file fname. If the "resid" of a residue has been set (either by using getpdb to create the molecule, or by an explicit operation in an nab routine) then columns 22-27 of the output pdb file will use it; otherwise, nab will assign a chain-id and residue number and use those. In this latter case, a molecule with a single strand will have a blank chain-id; if there is more than one strand, each strand is written as a separate chain with chain id "A" assigned to the first strand in mol, "B" to the second, etc. Options for putpdb are given in Table 36.2.

putbnd() writes the bonds of mol into fname. Each bond is a pair of integers on a line. The integers refer to atom records in the corresponding PDB-style file. putdist() writes the interatomic distances between all atoms of mol *a i , a j* where *i*<*j*, in this seven column format.

rnum1 rname1 aname1 rnum2 rname2 aname2 distance

36.16. Other Molecular Functions

matrix superimpose(molecule mol, string aex1, molecule r_mol, string aex2);
int rmsd(molecule mol, string aex1, molecule r_mol, string aex2, float r);
float angle(molecule mol, string aex1, string aex2, string aex3);
float anglep(point pt1, point pt2, point pt3);
float torsion(molecule mol, string aex1, string aex2, string aex3, string aex4);
float torsionp(point pt1, point pt2, point pt3, point pt4);
float dist(molecule mol, string aex1, string aex2);
float distp(point pt1, point pt2);
int countmolatoms(molecule mol, string aex);
int sugarpuckeranal(molecule mol, int strandnum, int startres, int endres);

keyword	*meaning*
-pqr	Put (or get) charges and radii into the columns following the xyz coordinates.
-nobocc	Do not put occupancy and b-factor into the columns following the xyz coordinates. This is implied if *-pqr* is present, but may also be used to save space in the output file, or for compatibility with programs that do not work well if such data is present.
-brook	Convert atom and residue names to the conventions used in Brookhaven PDB (version 2) files. This often gives greater compatibility with other software that may expect these conventions to hold, but the conversion may not be what is desired in many cases. Also, put the first character of the atom name in column 78, a preliminary effort at identifying it as in the most recent PDB format. If the *-brook* flag is not present, no conversion of atom and residue names is made, and no id is in column 78.
-wwpdb	Same as the -brook option, except use the "wwPDB" (aka version 3) residue and atom naming scheme.
-nocid	For *getpdb*, ignore the input chain id's (column 22 of PDB-format files), and generate strand names as consecutive integers. For *putpdb*, do not put the chain-id in the output (i.e., if this flag is present, the chain-id column will be blank). This can be useful when many water molecules are present.
-allcid	If set, create a chain ID for every strand in the molecule being written. Use the strand's name if it is an upper case letter, else use the next free upper case letter. Use a blank if no more upper case letters are available. Default is false.
-tr	Do not start numbering residues over again when a new chain is encountered, i.e., the residue numbers are consecutive across chains, as required by some force-field programs like Amber.

Table 36.2.: *Options for getpdb and putpdb.*

int helixanal(molecule mol);
int plane(molecule mol, string aex, float A, float B, float C);
float molsurf(molecule mol, string aex, float probe_rad);

superimpose() transforms molecule mol so that the root mean square deviation between corresponding atoms in mol and r_mol is minimized. The corresponding atoms are those selected by the atom expressions aex1 applied to mol and aex2 applied to r_mol. The atom expressions must select the same number of atoms in each molecule. No checking is done to insure that the atoms selected by the two atom expressions actually correspond. superimpose() returns the transformation matrix it found. rmsd() computes the root mean square deviation between the pairs of corresponding atoms selected by applying aex1 to mol and aex2 to r_mol and returns the value in r. The two atom expressions must select the same number of atoms. Again, it is the user's responsibility to insure the two atom expressions select corresponding atoms. rmsd() returns 0 on success and 1 on failure.

angle() and anglep() compute the angle in degrees between three points. angle() uses atoms expressions to determine the average coordinates of the sets. anglep() takes as an argument three explicit points. Similarly, torsion() and torsionp() compute a torsion angle in degrees defined by four points. torsion() uses atom expressions to specify the points. These atom expression match sets of atoms in mol. The points are defined by the average coordinates of the sets. torsionp() uses four explicit points. Both functions return 0 if the torsion angle is not defined.

dist() and distp() compute the distance in angstroms between two explicit atoms. dist() uses atom expressions to determine which atoms to include in the calculation. An atom expression which selects more than one atom results in the distance being calculated from the average coordinate of the selected atoms. distp() returns the distance between two explicit points. The function countmolatoms() returns the number of atoms selected by aex in mol.

sugarpuckeranal() is a function that reports the various torsion angles in a nucleic acid structure. helixanal() is an interactive helix analysis function based on the methods described by Babcock *et al.*.[570]

The plane() routine takes an atom expression aex and calculates the least-squares plane and returns the answer in the form z = Ax + By + C. It returns the number of atoms used to calculate the plane.

The molsurf() routine is an NAB adaptation of Paul Beroza's program of the same name. It takes coordinates and radii of atoms matching the atom expression *aex* in the input molecule, and returns the molecular surface area (the area of the solvent-excluded surface), in square angstroms. To compute the solvent-accessible area, add the probe radius to each atom's radius (using a for(a in m) loop), and call molsurf with a zero value for probe_rad.

36.17. Debugging Functions

nab provides the following builtin functions that allow the user to write the contents of various nab objects to an ASCII file. The file must be opened for writing before any of these functions are called.

int dumpmatrix(file, matrix mat);
int dumpbounds(file f, bounds b, int binary);
float dumpboundsviolations(file f, bounds b, int cutoff);
int dumpmolecule(file f, molecule mol, int dres, int datom, int dbond);
int dumpresidue(file f, residue res, int datom, int dbond);
int dumpatom(file f, residue res, int anum, int dbond);
int assert(condition);
int debug(expression(s));

dumpmatrix() writes the 16 float values of mat to the file f. The matrix is written as four rows of four numbers. dumpbounds() writes the distance bounds information contained in b to the file f using this eight column format:

atom-number1 atom-number2 lower upper

If binary is set to a nonzero value, equivalent information is written in binary format, which can save disk-space, and is much faster to read back in on subsequent runs.

dumpboundsviolations() writes all the bounds violations in the bounds object that are more than *cutoff*, and returns the bounds violation energy. dumpmolecule() writes the contents of mol to the file f. If dres is 1, then detailed residue information will also be written. If datom or dbond is 1, then detailed atom and/or bond information will be written. dumpresidue() writes the contents of residue res to the file f. Again if datom or dbond is 1, detailed information about that residue's atoms and bonds will be written. Finally dumpatom() writes the contents of the atom anum of residue res to the file f. If dbond is 1, bonding information about that atom is also written.

The assert() statement will evaluate the condition expression, and terminate (with an error message) if the expression is not true. Unlike the corresponding "C" language construct (which is a macro), code is generated at compile time to indicate both the file and line number where the assertion failed, and to parse the condition expression and print the values of subexpressions inside it. Hence, for a code fragment like:

```
i=20; MAX=17;
assert( i < MAX );
```

the error message will provide the assertion that failed, its location in the code, and the current values of "i" and "MAX". If the *-noassert* flag is set at compile time, assert statements in the code are ignored.

The debug() statement will evaluate and print a comma-separated expression list along with the source file(s) and line number(s). Continuing the above example, the statement

```
debug( i, MAX );
```

would print the values of "i" and "MAX" to *stdout*, and continue execution. If the *-nodebug* flag is set at compile time, debug statements in the code are ignored.

36.18. Time and date routines

NAB incorporates a few interfaces to time and date routines:

```
string date();
string timeofday();
string ftime( string fmt );
```

The date() routine returns a string in the format "03/08/1999", and the timeofday() routine returns the current time as "13:45:00". If you need access to more sophisticated time and date functions, the ftime() routine is just a wrapper for the standard C routine strftime, where the format string is used to determine what is output; see standard C documentation for how this works.

36.19. Computational resource consumption functions

NAB has a small number of functions to provide information about computational resources used during the run:

```
int mme_timer();
int mme_rism_max_memory();
```

mme_timer() provides tables of execution times for mme functions executed. It does not provide a complete summary nor does it include functions not in the mme family. It is, however, useful for identifying the most expensive routines. mme_rism_max_memory() reports the maximum amount of memory allocated during a 3D-RISM calculation.

37. NAB: Rigid-Body Transformations

This chapter describes NAB functions to create and manipulate molecules through a variety of rigid-body transformations. This capability, when combined with distance geometry (described in the next chapter) offers a powerful approach to many problems in initial structure generation.

37.1. Transformation Matrix Functions

nab uses 4×4 matrices to hold coordinate transformations. nab provides these functions to create transformation matrices.

matrix newtransform(float dx, float dy, float dz, float rx, float ry, float rz);
matrix rot4(molecule mol, string aex1, string aex2, float ang);
matrix rot4p(point p1, point p2, float angle);

newtransform() creates a 4×4 matrix that will rotate an object by rz degrees about the Z axis, ry degrees about the Y axis, rx degrees about the X axis and then translate the rotated object by dx, dy, dz along the X, Y and Z axes. All rotations and transformations are with respect the standard X, Y and Z axes centered at (0,0,0). rot4() and rot4p() create transformation matrices that rotate an object about an arbitrary axis. The rotation amount is in degrees. rot4() uses two atom expressions to define an axis that goes from aex1 to aex2. If an atom expression matches more that one atom in mol, the average of the coordinates of the matched atoms are used. If an atom expression matches no atoms in mol, the zero matrix is returned. rot4p() uses explicit points instead of atom expressions to specify the axis. If p1 and p2 are the same, the zero matrix is returned.

37.2. Frame Functions

Every nab molecule has a "frame" which is three orthonormal vectors and their origin. The frame acts like a handle attached to the molecule allowing control over its movement. Two frames attached to different molecules allow for precise positioning of one molecule with respect to the other. These functions are used in frame creation and manipulation. All return 0 on success and 1 on failure.

int setframe(int use, molecule mol, string org, string xtail, string xhead,
string ytail, string yhead);
int setframep(int use, molecule mol, point org, point xtail, point xhead,
point ytail, point yhead);
int alignframe(molecule mol, molecule r_mol);

setframe() and setframep() create coordinate frames for molecule mol from an origin and two independent vectors. In setframe(), the origin and two vectors are specified by atom expressions. These atom expressions match sets of atoms in mol. The average coordinates of the selected sets are used to define the origin (org), an X-axis (xtail to xhead) and a Y-axis (ytail to yhead). The Z-axis is created as X×Y. Since it is unlikely that the original X and Y axes are orthogonal, the parameter use specifies which of them is to be a real axis. If use == 1, then the specified X-axis is the real X-axis and Y is recreated from Z×X. If use == 2, then the specified Y-axis is the real Y-axis and X is recreated from Y×Z. setframep() works exactly the same way except the vectors and origin are specified as explicit points.

alignframe() transforms mol to superimpose its frame on the frame of r_mol. If r_mol is NULL, alignframe() transforms mol to superimpose its frame on the standard X,Y,Z directions centered at (0,0,0).

37.3. Functions for working with Atomic Coordinates

nab provides several functions for getting and setting user defined sets of molecular coordinates.

int setpoint(molecule mol, string aex, point pt);
int setxyz_from_mol(molecule mol, string aex, point pts[]);
int setxyzw_from_mol(molecule mol, string aex, float xyzw[]);
int setmol_from_xyz(molecule mol, string aex, point pts[]);
int setmol_from_xyzw(molecule mol, string aex, float xyzw[]);
int transformmol(matrix mat, molecule mol, string aex);
residue transformres(matrix mat, residue res, string aex);

setpoint() sets pt to the average value of the coordinates of all atoms selected by the atom expression aex. If no atoms were selected it returns 1, otherwise it returns a 0. setxyz_from_mol() copies the coordinates of all atoms selected by the atom expression aex to the point array pt. It returns the number of atoms selected. setmol_from_xyz() replaces the coordinates of the selected atoms from the values in pt. It returns the number of replaced coordinates. The routines setxyzw_from_mol and setmol_from_xyzw work in the same way, except that they use four-dimensional coordinates rather than three-dimensional sets.

transformmol() applies the transformation matrix mat to those atoms of mol that were selected by the atom expression aex. It returns the number of atoms selected. transformres() applies the transformation matrix mat to those atoms of res that were selected by the atom expression aex and returns a transformed *copy* of the input residue. It returns NULL if the operation failed.

37.4. Symmetry Functions

Here we describe a set of NAB routines that provide an interface for rigid-body transformations based on crystallographic, point-group, or other symmetries. These are primarily higher-level ways to creating and manipulating sets of transformation matrices corresponding to common types of symmetry operations.

37.4.1. Matrix Creation Functions

int MAT_cube(point pts[3], matrix mats[24])
int MAT_ico(point pts[3], matrix mats[60])
int MAT_octa(point pts[3], matrix mats[24])
int MAT_tetra(point pts[3], matrix mats[12])
int MAT_dihedral(point pts[3], int nfold, matrix mats[1])
int MAT_cyclic(point pts[2], float ang, int cnt, matrix mats[1])
int MAT_helix(point pts[2], float ang, float dst, int cnt, matrix mats[1])
int MAT_orient(point pts[4], float angs[3], matrix mats[1])
int MAT_rotate(point pts[2], float ang, matrix mats[1])
int MAT_translate(point pts[2], float dst, matrix mats[1])

These two groups of functions produce arrays of matrices that can be applied to objects to generate point group symmetries (first group) or useful transformations (second group). The operations are defined with respect to a center and a set of axes specified by the points in the array pts[]. Every function requires a center and one axis which are pts[1] and the vector pts[1]→pts[2]. The other two points (if required) define two additional directions: pts[1]→pts[3] and pts[1]→pts[4]. How these directions are used depends on the function.

The point groups generated by the functions MAT_cube(), MAT_ico(), MAT_octa() and MAT_tetra() have three internal 2-fold axes. While these 2-fold are orthogonal, the 2 directions specified by the three points in pts[] need only be independent (not parallel). The 2-fold axes are constructed in this fashion. Axis-1 is along the direction pts[1]→pts[2]. Axis-3 is along the vector pts[1]→pts[2] × pts[1]→pts[3] and axis-2 is recreated along the vector axis-3 × axis-1. Each of these four functions creates a fixed number of matrices.

Dihedral symmetry is generated by an N-fold rotation about an axis followed by a 2-fold rotation about a second axis orthogonal to the first axis. MAT_dihedral() produces matrices that generate this symmetry. The N-fold axis is pts[0]→pts[1] and the second axis is created by the same orthogonalization process described above. Unlike the previous point group functions the number of matrices created by MAT_dihedral() is not fixed but is equal to $2 \times nfold$.

MAT_cyclic() creates cnt matrices that produce uniform rotations about the axis pts[1]→pts[2]. The rotations are in multiples of the angle ang beginning with o, and increasing by ang until cnt matrices have been created. cnt is required to be > 0, but ang can be 0, in which case MAT_cyclic returns cnt copies of the identity matrix.

MAT_helix() creates cnt matrices that produce a uniform helical twist about the axis pts[1]→pts[2]. The rotations are in multiples of ang and the translations in multiples of dst. cnt must be > 0, but either ang or dst or both may be zero. If ang is not 0, but dst is, MAT_helix() produces a uniform plane rotation and is equivalent to MAT_cyclic(). An ang of 0 and a nonzero dst produces matrices that generate a uniform translation along the axis. If both ang and dst are 0, the MAT_helix() creates cnt copies of the identity matrix.

The three functions MAT_orient(), MAT_rotate() and MAT_translate() are not really symmetry operations but are auxiliary operations that are useful for positioning the objects which are to be operated on by the true symmetry operators. Two of these functions MAT_rotate() and MAT_translate() produce a single matrix that either rotates or translates an object along the axis pts[1]→pts[2]. A zero ang or dst is acceptable in which case the function creates an identity matrix. Except for a different user interface these two functions are equivalent to the nab builtins rot4p() and tran4p().

MAT_orient() creates a matrix that rotates a object about the three axes pts[1]→pts[2], pts[1]→pts[3] and pts[1]→pts[4]. The rotations are specified by the values of the array angs[], with ang[1] the rotation about axis-1 etc. The rotations are applied in the order axis-3, axis-2, axis-1. The axes remained fixed throughout the operation and zero angle values are acceptable. If all three angles are zero, MAT_orient() creates an identity matrix.

37.4.2. Matrix I/O Functions

int MAT_fprint(file f, int nmats, matrix mats[1])
int MAT_sprint(string str, int nmats, matrix mats[1])
int MAT_fscan(file f, int smats, matrix mats[1])
int MAT_sscan(string str, int smats, matrix mats[1])
string MAT_getsyminfo()

This group of functions is used to read and write nab matrix variables. The two functions MAT_fprint() and MAT_sprint() write the the matrix to the file f or the string str. The number of matrices is specified by the parameter nmats and the matrices are passed in the array mats[].

The two functions MAT_fscan() and MAT_sscan() read matrices from the file f or the string str into the array mats[]. The parameter smats is the size of the matrix array and if the source file or string contains more than smats only the first smats will be returned. These two functions return the number of matrices read unless there the number of matrices is greater than smat or the last matrix was incomplete in which case they return -1.

In order to understand the last function in this group, MAT_getsyminfo(), it is necessary to discuss both the internal structure the nab matrix type and one of its most important uses. The nab matrix type is used to hold transformation matrices. Although these are atomic objects at the nab level, they are actually 4×4 matrices where the first three elements of the fourth row are the X Y and Z components of the translation part of the transformation. The matrix print functions write each matrix as four lines of four numbers separated by a single space. Similarly the matrix read functions expect each matrix to be represented as four lines of four white space (any number of tabs and spaces) separated numbers. The print functions use %13.6e for each number in order to produce output with aligned columns, but the scan functions only require that each matrix be contained in four lines of four numbers each.

Most nab programs use matrix variables as intermediates in creating structures. The structures are then saved and the matrices disappear when the program exits. Recently nab was used to create a set of routines called a “symmetry server”. This is a set of nab programs that work together to create matrix streams that are used to assemble composite objects. In order to make it most general, the symmetry server produces only matrices leaving it to the user to apply them. Since these programs will be used to create hierarchies of symmetries or transformations we decided that the external representation (files or strings) of matrices would consist of two kinds of information — required lines of row values and optional lines beginning with the character # some of which are used to contain information that describes how these matrices were created.

MAT_getsyminfo() is used to extract this symmetry information from either a matrix file or a string that holds the contents of a matrix file. Each time the user calls MAT_fscan() or MAT_sscan(), any symmetry information present in the source file or string is saved in private buffer. The previous contents of this buffer are overwritten and lost. MAT_getsyminfo() returns the contents of this buffer. If the buffer is empty, indicating no symmetry information was present in either the source file or string, MAT_getsyminfo() returns NULL.

37.5. Symmetry server programs

This section describes a set of nab programs that are used together to create composite objects described by a hierarchical nest of transformations. There are four programs for creating and operating on transformation matrices: matgen, matmerge, matmul and matextract, a program, transform, for transforming PDB or point files, and two programs, tss_init and tss_next for searching spaces defined by transformation hierarchies. In addition to these programs, all of this functionality is available directly at the nab level via the MAT_ and tss_ builtins described above.

37.5.1. matgen

The program matgen creates matrices that correspond to a symmetry or transformation operation. It has one required argument, the name of a file containing a description of this operation. The created matrices are written to stdout. A single matgen may be used by itself or two or more matgen programs may be connected in a pipeline producing nested symmetries.

```
matgen -create sydef-1 | matgen symdef-2 | ... | matgen symdef-N
```

Because a matgen can be in the middle of a pipeline, it automatically looks for an stream of matrices on stdin. This means the first matgen in a pipeline will wait for an EOF (generally Ctl-D) from the terminal unless connected to an empty file or equivalent. In order to avoid the nuisance of having to create an empty matrix stream the first matgen in a pipeline should use the -create flag which tells matgen to ignore stdin.

If input matrices are read, each input matrix left multiplies the first generated matrix, then the second etc. The table below shows the effect of a matgen performing a 2-fold rotation on an input stream of three matrices.

Input:	IM_1, IM_2, IM_3
Operation:	2-fold rotation: R_1, R_2
Output:	$IM_1 \times R_1$, $IM_2 \times R_1$, $IM_3 \times R_1$, $IM_1 \times R_2$, $IM_2 \times R_2$, $IM_3 \times R_2$

37.5.2. Symmetry Definition Files

Transformations are specified in text files containing several lines of keyword/value pairs. These lines define the operation, its associated axes and other parameters such as angles, a distance or count. Most keywords have a default value, although the operation, center and axes are always required. Keyword lines may be in any order. Blank lines and most lines starting with a sharp (#) are ignored. Lines beginning with #S{, #S+ and #S} are structure comments that describe how the matrices were created. These lines are required to search the space defined by the transformation hierarchy and their meaning and use is covered in the section on “Searching Transformation Spaces”. A complete list of keywords, their acceptable values and defaults is shown below.

Keyword	Default Value	Possible Values
symmetry	None	cube, cyclic, dihedral, dodeca, helix, ico, octa, tetra.
transform	None	orient, rotate, translate.
name	mPid	Any string of nonblank characters.
noid	false	true, false.
axestype	relative	absolute, relative.
center	None	Any three numbers separated by tabs or spaces.
axis, axis1	None	
axis2	None	
axis3	None	
angle,angle1	0	Any number.
angle2	0	
angle3	0	
dist	0	
count	1	Any integer.

axis and axis1 are synonyms as are angle and angle1.

The symmetry and transform keywords specify the operation. One or the other but not both must be specified.

The name keyword names a particular symmetry operation. The default name is m immediately followed by the process ID, eg m2286. name is used by the transformation space search routines tss_init and tss_next and is described later in the section "Searching Transformation Spaces".

The noid keyword with value true suppresses generation of the identity matrix in symmetry operations. For example, the keywords below

```
symmetry cyclic
noid false
center 0 0 0
axis 0 0 1
count 3
```

produce three matrices which perform rotations of 0o, 120o and 240o about the Z-axis. If noid is true, only the two non-identity matrices are created. This option is useful in building objects with two or three orthogonal 2-fold axes and is discussed further in the example "Icosahedron from Rotations". The default value of noid is false.

The axestype, center and axis* keywords defined the symmetry axes. The center and axis* keywords each require a point value which is three numbers separated by tabs or spaces. Numbers may integer or real and in fixed or exponential format. Internally all numbers are converted to nab type float which is actually double precision. No space is permitted between the minus sign of a negative number and the digits.

The interpretation of these points depends on the value of the keyword axestype. If it is absolute then the axes are defined as the vectors center→axis1, center→axis2 and center→axis3. If it relative, then the axes are vectors whose directions are **O→axis1**, **O→axis2** and **O→axis3** with their origins at center. If the value of center is 0,0,0, then absolute and relative are equivalent. The default value axestype is relative; center and the axis* do not have defaults.

The angle keywords specify the rotation about the axes. angle1 is associated with axis1 etc. Note that angle and angle1 are synonyms. The angle is in degrees, with positive being in the counterclockwise direction as you sight from the axis point to the center point. Either an integer or real value is acceptable. No space is permitted between the minus sign of a negative number and its digits. All angle* keywords have a default value of 0.

The dist keyword specifies the translation along an axis. The positive direction is from center to axis. Either integer or real value is acceptable. No space is permitted between the minus sign of a negative number and its digits. The default value of dist is 0.

The count keyword is used in three related ways. For the cyclic value of the symmetry it specifies ount matrices, each representing a rotation of 360/count. It also specifies the same rotations about the non 2-fold axis of dihedral symmetry. For helix symmetry, it indicates that count matrices should be created, each with a rotation of angle. In all cases the default value is 1.

This table shows which keywords are used and/or required for each type of operation.

symmetry	**name**	**noid**	**axestype**	**center**	***axes***	***angles***	**dist**	**count**
cube	m*Pid*	false	relative	Required	1,2	-	-	-
cyclic	m*Pid*	false	relative	Required	1	-	-	D=1
dihedral	m*Pid*	false	relative	Required	1,2	-	-	D=1
dodeca	m*Pid*	false	relative	Required	1,2	-	-	-
helix	m*Pid*	false	relative	Required	1	1,D=0	D=0	D=1
ico	m*Pid*	false	relative	Required	1,2	-	-	-
octa	m*Pid*	false	relative	Required	1,2	-	-	-
tetra	m*Pid*	false	relative	Required	1,2	-	-	-

transform	**name**	**noid**	**axestype**	**center**	***axes***	***angles***	**dist**	**count**
orient	m*Pid*	-	relative	Required	All	All,D=0	-	-
rotate	m*Pid*	-	relative	Required	1	1,D=0	-	-
translate	m*Pid*	-	relative	Required	1	-	D=0	-

37.5.3. matmerge

The matmerge program combines 2-4 files of matrices into a single stream of matrices written to stdout. Input matrices are in files whose names are given on as arguments on the matmerge command line. For example, the command line below

```
matmerge A.mat B.mat C.mat
```

copies the matrices from A.mat to stdout, followed by those of B.mat and finally those of C.mat. Thus matmerge is similar to the Unix cat command. The difference is that while they are called matrix files, they can contain special comments that describe how the matrices they contain were created. When matrix files are merged, these comments must be collected and grouped so that they are kept together in any further matrix processing.

37.5.4. matmul

The matmul program takes two files of matrices, and creates a new stream of matrices formed by the pair wise product of the matrices in the input streams. The new matrices are written to stdout. If the number of matrices in the two input files differ, the last matrix of the shorter file is replicated and applied to all remaining matrices of the longer file. For example, if the file 3.mat has three matrices and the file 5.mat has five, then the command "matmul 3.mat 5.mat" would result in the third matrix of 3.mat multiplying the third, forth and fifth matrices of 5.mat.

37.5.5. matextract

The matextract is used to extract matrices from the matrix stream presented on stdin and writes them to stdout. Matrices are numbered from 1 to N, where N is the number of matrices in the input stream. The matrices are selected by giving their numbers as the arguments to the matextract command. Each argument is comma or space separated list of one or more ranges, where a range is either a number or two numbers separated by a dash (-). A range beginning with - starts with the first matrix and a range ending with - ends with the last matrix. The range - selects all matrices. Here are some examples.

Command	**Action**
matextract 2	Extract matrix number 2.
matextract 2,5	Extract matrices number 2 and 5.
matextract 2 5	Extract matrices number 2 and 5.
matextract 2-5	Extract matrices number 2 up to and including 5.
matextract -5	Extract matrices 1 to 5.
matextract 2-	Extract all matrices beginning with number 2.
matextract -	Extract all matrices.
matextract 2-4,7 13 15,19-	Extract matrices 2 to 4, 7, 13, 15 and all matrices numbered 19 or higher.

37.5.6. transform

The `transform` program applies matrices to an object creating a composite object. The matrices are read from `stdin` and the new object is written to `stdout`. `transform` takes one argument, the name of the file holding the object to be transformed. `transform` is limited to two types of objects, a molecule in PDB format, or a set of points in a text file, three space/tab separated numbers/line. The name of object file is preceded by a flag specifying its type.

Command	Action
`transform -pdb X.pdb`	Transform a PDB format file.
`transform -point X.pts`	Transform a set of points.

38. NAB: Distance Geometry

The second main element in NAB for the generation of initial structures is distance geometry. The next subsection gives a brief overview of the basic theory, and is followed by sections giving details about the implementation in NAB.

38.1. Metric Matrix Distance Geometry

A popular method for constructing initial structures that satisfy distance constraints is based on a metric matrix or "distance geometry" approach.[642, 652] If we consider describing a macromolecule in terms of the distances between atoms, it is clear that there are many constraints that these distances must satisfy, since for N atoms there are $N(N-1)/2$distances but only $3N$ coordinates. General considerations for the conditions required to "embed" a set of interatomic distances into a realizable three-dimensional object forms the subject of distance geometry. The basic approach starts from the *metric matrix* that contains the scalar products of the vectors $\mathbf{x}_i$ that give the positions of the atoms:

$$g_{ij} \equiv \mathbf{x}_i \cdot \mathbf{x}_j \tag{38.1}$$

These matrix elements can be expressed in terms of the distances d_{ij}:

$$g_{ij} = 2(d_{i0}^2 + d_{j0}^2 - d_{ij}^2) \tag{38.2}$$

If the origin ("0") is chosen at the centroid of the atoms, then it can be shown that distances from this point can be computed from the interatomic distances alone. A fundamental theorem of distance geometry states that a set of distances can correspond to a three-dimensional object only if the metric matrix **g** is rank three, i.e., if it has three positive and N-3 zero eigenvalues. This is not a trivial theorem, but it may be made plausible by thinking of the eigenanalysis as a principal component analysis: all of the distance properties of the molecule should be describable in terms of three "components," which would be the x, y and z coordinates. If we denote the eigenvector matrix as **w** and the eigenvalues λ, the metric matrix can be written in two ways:

$$g_{ij} = \sum_{k=1}^{3} x_{ik}x_{jk} = \sum_{k=1}^{3} w_{ik}w_{jk}\lambda_k \tag{38.3}$$

The first equality follows from the definition of the metric tensor, Eq. (1); the upper limit of three in the second summation reflects the fact that a rank three matrix has only three nonzero eigenvalues. Eq. (3) then provides an expression for the coordinates $\mathbf{x}_i$ in terms of the eigenvalues and eigenvectors of the metric matrix:

$$x_{ik} = \lambda_k^{1/2} w_{ik} \tag{38.4}$$

If the input distances are not exact, then in general the metric matrix will have more than three nonzero eigenvalues, but an approximate scheme can be made by using Eq. (4) with the three largest eigenvalues. Since information is lost by discarding the remaining eigenvectors, the resulting distances will not agree with the input distances, but will approximate them in a certain optimal fashion. A further "refinement" of these structures in three-dimensional space can then be used to improve agreement with the input distances.

In practice, even approximate distances are not known for most atom pairs; rather, one can set upper and lower bounds on acceptable distances, based on the covalent structure of the protein and on the observed NOE cross peaks. Then particular instances can be generated by choosing (often randomly) distances between the upper and lower bounds, and embedding the resulting metric matrix.

Considerable attention has been paid recently to improving the performance of distance geometry by examining the ways in which the bounds are "smoothed" and by which distances are selected between the bounds.[653, 654] The use of triangle bound inequalities to improve consistency among the bounds has been used for many years, and NAB implements the "random pairwise metrization" algorithm developed by Jay Ponder.[644] Methods like these are important especially for underconstrained problems, where a goal is to generate a reasonably random distribution of acceptable structures, and the difference between individual members of the ensemble may be quite large.

An alternative procedure, which we call "random embedding", implements the procedure of deGroot *et al.* for satisfying distance constraints.[655] This does not use the embedding idea discussed above, but rather randomly corrects individual distances, ignoring all couplings between distances. Doing this a great many times turns out to actually find fairly good structures in many cases, although the properties of the ensembles generated for underconstrained problems are not well understood. A similar idea has been developed by Agrafiotis,[656] and we have adopted a version of his "learning parameter" strategy into our implementation.

Although results undoubtedly depend upon the nature of the problem and the constraints, in many (most?) cases, randomized embedding will be both faster and better than the metric matrix strategy. Given its speed, randomized embedding should generally be tried first.

38.2. Creating and manipulating bounds, embedding structures

A variety of metric-matrix distance geometry routines are included as builtins in nab.

```
bounds newbounds( molecule mol, string opts );
int andbounds( bounds b, molecule mol, string aex1, string aex2,
        float lb, float ub );
int orbounds( bounds b, molecule mol, string aex1, string aex2,
        float lb, float ub );
int setbounds( bounds b, molecule mol, string aex1, string aex2,
        float lb, float ub );
int showbounds( bounds b, molecule mol, string aex1, string aex2 );
int useboundsfrom( bounds b, molecule mol1, string aex1, molecule mol2,
            string aex2, float deviation );
int setboundsfromdb( bounds b, molecule mol, string aex1, string aex2,
            string dbase, float mul );
int setchivol( bounds b, molecule mol, string aex1, string aex2, string aex3, string aex4, float vol );
int setchiplane( bounds b, molecule mol, string aex );
float getchivol( molecule mol, string aex1, string aex2, string aex3, string aex4 );
float getchivolp( point p1, point p2, point p3, point p4 );
int tsmooth( bounds b, float delta );
int geodesics( bounds b );
int dg_options( bounds b, string opts );
int embed( bounds b, float xyz[] );
```

The call to newbounds() is necessary to establish a bounds matrix for further work. This routine sets lower bounds to van der Waals limits, along with bounds derived from the input geometry for atoms bonded to each other, and for atoms bonded to a common atoms (i.e., so-called 1-2 and 1-3 interactions.) Upper and lower bounds for 1-4 interactions are set to the maximum and minimum possibilities (the max (*syn* , "van der Waals limits") and *anti* distances). newbounds() has a string as its last parameter. This string is used to pass in options that control the details of how those routines execute. The string can be NULL, "" or contain one or more *option*s surrounded by white space. The formats of an option are

```
-name=value
-name to select the default value if it exists.
```

The options to newbounds() are listed in Table 38.1.

Option	type	Default	Action
-rbm	string	None	The value of the option is the name of a file containing the bounds matrix for this molecule. This file would ordinarily be made by the dump-bounds command.
-binary			If this flag is present, bounds read in with the *-rbm* will expect a binary file created by the dumpbounds command.
-nocov			If this flag is present, no covalent (bonding) information will be used in constructing the bounds matrix.
-nchi	int	4	The option containing the keyword *nchi* allocates *n* extra chiral atoms for each residue of this molecule. This allows for additional chirality information to be provided by the user. The default is 4 extra chiral atoms per residue.

Table 38.1.: *Options to newbounds.*

The next five routines use atom expressions aex1 and aex2 to select two sets of atoms. Each of these four routines returns the number of bounds set or changed. For each pair of atoms ($a1$ in aex1 and $a2$ in aex2) andbounds() sets the lower bound to max (*current_lb, lb*) and the upper bound to the min (*current_ub, ub*). If $ub < current\_lb$ or if $lb > current\_ub$, the bounds for that pair are unchanged. The routine orbounds() works in a similar fashion, except that it uses the less restrictive of the two sets of bounds, rather than the more restrictive one. The setbounds() call updates the bounds, overwriting whatever was there. showbounds() prints all the bounds between the atoms selected in the first atom expression and those selected in the second atom expression. The useboundsfrom() routine sets the the bounds between all the selected atoms in *mol1* according to the geometry of a reference molecule, *mol2*. The bounds are set between every pair of atoms selected in the first atom expression, *aex1* to the distance between the corresponding pair of atoms selected by *aex2* in the reference molecule. In addition, a slack term, *deviation*, is used to allow some variance from the reference geometry by decreasing the lower bound and increasing the upper bound between every pair of atoms selected. The amount of increase or decrease depends on the distance between the two atoms. Thus, a *deviation* of 0.25 will result in the lower bound set between two atoms to be 75% of the actual distance separating the corresponding two atoms selected in the reference molecule. Similarly, the upper bound between two atoms will be set to 125% of the actual distance separating the corresponding two atoms selected in the reference molecule. For instance, the call

```
useboundsfrom(b, mol1, "1:2:C1',N1", mref, "3:4:C1',N1", 0.10 );
```

sets the lower bound between the C1’ and N1 atoms in strand 1, residue 2 of molecule *mol1* to 90% of the distance between the corresponding pair of atoms in strand 3, residue 4 of the reference molecule, *mref*. Similarly, the upper bound between the C1’ and N1 atoms selected in *mol1* is set to 110% of the distance between the corresponding pair of atoms in *mref*. A *deviation* of 0.0 sets the upper and lower bounds between every pair of atoms selected to be the actual distance between the corresponding reference atoms. If *aex1* selects the same atoms as *aex2*, the bounds between those atoms selected will be constrained to the current geometry. Thus the call,

```
useboundsfrom(b, mol1, "1:1:", mol1, "1:1", 0.0 );
```

essentially constrains the current geometry of all the atoms in strand 1, residue 1, by setting the upper and lower bounds to the actual distances separating each atom pair. useboundsfrom() only checks the number of atoms selected by *aex1* and compares it to the number of atoms selected by *aex2*. If the number of atoms selected by both atom expressions are not equal, an error message is output. Note, however, that there is no checking on the atom types selected by either atom expression. Hence, it is important to understand the method in which nab atom expressions are evaluated. For more information, refer to Section 2.6, “Atom Names and Atom Expressions”.

The useboundsfrom() function can also be used with distance geometry "templates", as discussed in the next subsection.

The routine setchivol() uses four atom expressions to select exactly four different atoms and sets the volume of the chiral (ordered) tetrahedron they describe to vol. Setting vol to 0 forces the four atoms to be planar. setchivol()

returns 0 on success and 1 on failure. setchivol() does not affect any distance bounds in b and may precede or follow triangle smoothing.

Similar to setchivol(), setchiplane() enforces planarity across four or more atoms by setting the chiral volume to 0 for every quartet of atoms selected by aex. setchiplane() returns the number of quartets constrained. Note: If the number of chiral constraints set is larger than the default number of chiral objects allocated in the call to newbounds(), a chiral table overflow will result. Thus, it may be necessary to allocate space for additional chiral objects by specifying a larger number for the option *nchi* in the call to newbounds().

getchivol() takes as an argument four atom expressions and returns the chiral volume of the tetrahedron described by those atoms. If more than one atom is selected for a particular point, the atomic coordinate is calculated from the average of the atoms selected. Similarly, getchivolp() takes as an argument four parameters of type point and returns the chiral volume of the tetrahedron described by those points.

After bounds and chirality have been set in this way, the general approach would be to call tsmooth() to carry out triangle inequality smoothing, followed by embed() to create a three-dimensional object. This might then be refined against the distance bounds by a conjugate-gradient minimization routine. The tsmooth() routine takes two arguments: a bounds object, and a tolerance parameter *delta*, which is the amount by which an upper bound may exceed a lower bound without triggering a triangle error. For most circumstances, *delta* would be chosen as a small number, like 0.0005, to allow for modest round-off. In some circumstances, however, *delta* could be larger, to allow some significant inconsistencies in the bounds (in the hopes that the problems would be fixed in subsequent refinement steps.) If the tsmooth() routine detects a violation, it will (arbitrarily) adjust the upper bound to equal the lower bound. Ideally, one should fix the bounds inconsistencies before proceeding, but in some cases this fix will allow the refinements to proceed even when the underlying cause of the inconsistency is not corrected.

For larger systems, the tsmooth() routine becomes quite time-consuming as it scales O(ˆ3). In this case, a more efficient triangle smoothing routine, geodesics() is used. geodesics() smoothes the bounds matrix via the triangle inequality using a sparse matrix version of a shortest path algorithm.

The embed routine takes a bounds object as input, and returns a four-dimensional array of coordinates; (values of the 4-th coordinate may be nearly zero, depending on the value of *k*4*d*, see below.) Options for how the embed is done are passed in through the *dg_options* routine, whose option string has *name*=*value* pairs, separated by commas or whitespace. Allowed options are listed in the following table.

keyword	*default*	*meaning*
ddm	none	Dump distance matrix to this file.
rdm	none	Instead of creating a distance matrix, read it from this file.
dmm	none	Dump the metric matrix to this file.
rmm	none	Instead of creating a metric matrix, read it from this file.
gdist	0	If set to nonzero value, use a Gaussian distribution for selecting distances; this will have a mean at the center of the allowed range, and a standard deviation equal to 1/4 of the range. If gdist=0, select distances from a uniform distribution in the allowed range.
randpair	0	Use random pair-wise metrization for this percentage of the distances, i.e., randpair=10. would metrize 10% of the distance pairs.
eamax	10	Maximum number of embed attempts before bailing out.
seed	-1	Initial seed for the random number generator.
pembed	0	If set to a nonzero value, use the "proximity embedding" scheme of de Groot *et al.*, [26] and Agrafiotis [27], rather than metric matrix embedding.
shuffle	1	Set to 1 to randomize coordinates inside a box of dimension *rbox* at the beginning of the *pembed* scheme; if 0, use whatever coordinates are fed to the routine.
rbox	20.0	Size, in angstroms, of each side of the cubic into which the coordinates are randomly created in the proximity-embed procedure, if *shuffle* is set.

keyword	*default*	*meaning*
riter	1000	Maximum number of cycles for random-embed procedure. Each cycle selects 1000 pairs for adjustment.
slearn	1.0	Starting value for the learning parameter in proximity embedding; see [27] for details.
kchi	1.0	Force constant for enforcement of chirality constraints.
k4d	1.0	Force constant for squeezing out the fourth dimensional coordinate. If this is nonzero, a penalty function will be added to the bounds-violation energy, which is equal to 0.5 * *k4d* * *w* * *w*, where *w* is the value of the fourth dimensional coordinate.
sqviol	0	If set to nonzero value, use parabolas for the violation energy when upper or lower bounds are violated; otherwise use functions based on those in the dgeom program. See the code in embed.c for details.
lbpen	3.5	Weighting factor for lower-bounds violations, relative to upper-bounds violations. The default penalizes lower bounds 3.5 times as much as the equivalent upper-bounds violations, which is frequently appropriate distance geometry calculations on molecules.
ntpr	10	Frequency at which the bounds matrix violations will be printed in subsequent refinements.
pencut	-1.0	If pencut >= 0.0, individual distance and chirality violations greater than pencut will be printed out (along with the total energy) every ntpr steps.

Typical calling sequences. The following segment shows some ways in which these routines can be put together to do some simple embeds:

```
1  molecule m;
2  bounds b;
3  float fret, xyz[ 10000 ];
4  int ier;
5
6  m = getpdb( argv[2] );
7  b = newbounds( m, "" );
8  tsmooth( b, 0.0005 );
9
10 dg_options( b, "gdist=1, ntpr=50, k4d=2.0, randpair=10." );
11 embed( b, xyz );
12 ier = conjgrad( xyz, 4*m.natoms, fret, db_viol, 0.1, 10., 200 );
13 printf( "conjgrad returns %d\\n", ier );
14
15 setmol_from_xyzw( m, NULL, xyz );
16 putpdb( "new.pdb", m );
```

In lines 6-8, the molecule is created by reading in a pdb file, then bounds are created and smoothed for it. The embed options (established in line 10) include 10% random pairwise metrization, use of Gaussian distance selection, squeezing out the 4-th dimension with a force constant of 2.0, and printing every 50 steps. The coordinates developed in the *embed* step (line 11) are passed to a conjugate gradient minimizer (see the description below), which will minimize for 200 steps, using the bounds-violation routine *db_viol* as the target function. Finally, in lines 15-16, the *setmol_from_xyzw* routine is used to put the coordinates from the *xyz* array back into the molecule, and a new pdb file is written.

More complex and representative examples of distance geometry are given in the **Examples** chapter below.

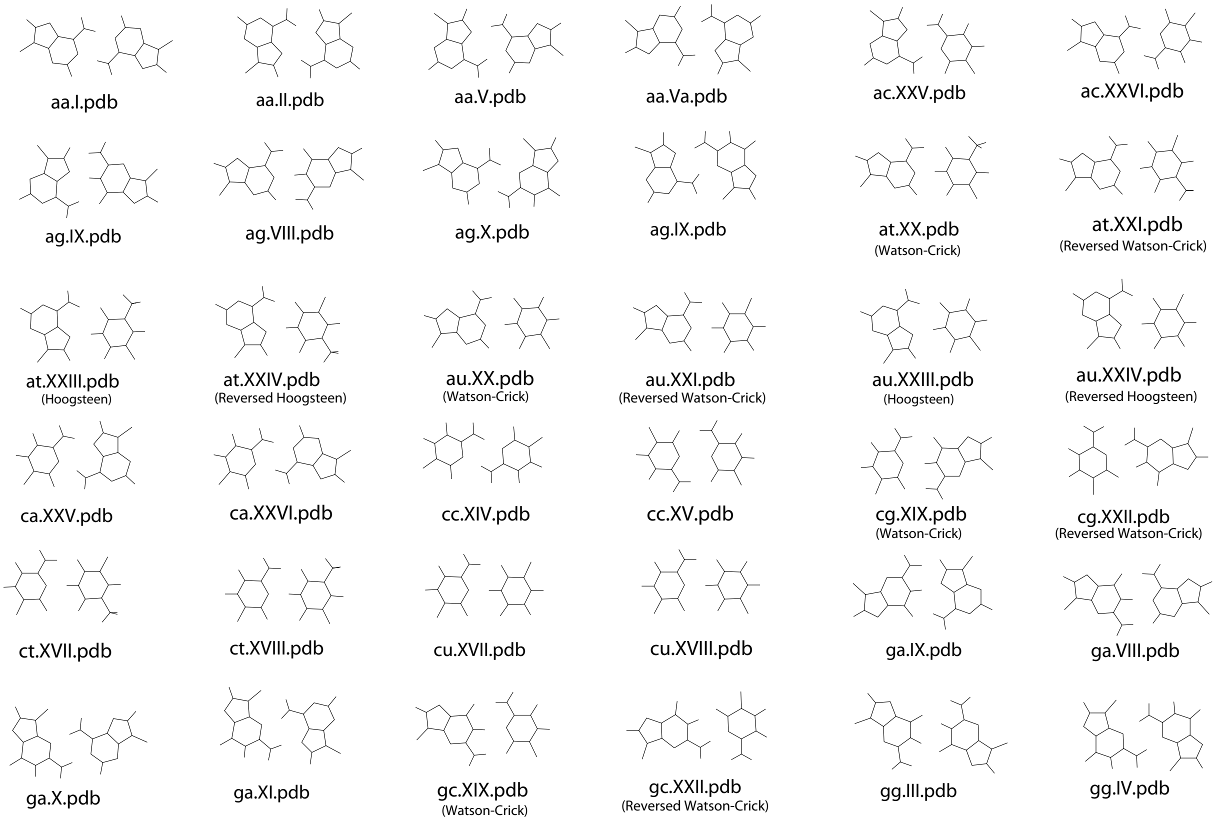

aa.I.pdb
aa.II.pdb
aa.V.pdb
aa.Va.pdb
ac.XXV.pdb
ac.XXVI.pdb
ag.IX.pdb
ag.VIII.pdb
ag.X.pdb
ag.IX.pdb
at.XX.pdb
(Watson-Crick)
at.XXI.pdb
(Reversed Watson-Crick)
at.XXIII.pdb
(Hoogsteen)
at.XXIV.pdb
(Reversed Hoogsteen)
au.XX.pdb
(Watson-Crick)
au.XXI.pdb
(Reversed Watson-Crick)
au.XXIII.pdb
(Hoogsteen)
au.XXIV.pdb
(Reversed Hoogsteen)
ca.XXV.pdb
ca.XXVI.pdb
cc.XIV.pdb
cc.XV.pdb
cg.XIX.pdb
(Watson-Crick)
cg.XXII.pdb
(Reversed Watson-Crick)
ct.XVII.pdb
ct.XVIII.pdb
cu.XVII.pdb
cu.XVIII.pdb
ga.IX.pdb
ga.VIII.pdb
ga.X.pdb
ga.XI.pdb
gc.XIX.pdb
(Watson-Crick)
gc.XXII.pdb
(Reversed Watson-Crick)
gg.III.pdb
gg.IV.pdb

Figure 38.1.: *Basepair templates for use with usebounds from(), (aa-gg)*

38.3. Distance geometry templates

The useboundsfrom() function can be used with structures supplied by the user, or by canonical structures supplied with the nab distribution called "templates". These templates include stacking schemes for all standard residues in a A-DNA, B-DNA, C-DNA, D-DNA, T-DNA, Z-DNA, A-RNA, or A'-RNA stack. Also included are the 28 possible basepairing schemes as described in Saenger.[657] The templates are in PDB format and are located in $AMBERHOME/dat/dgdb/basepairs/ and $AMBERHOME/dat/dgdb/stacking/.

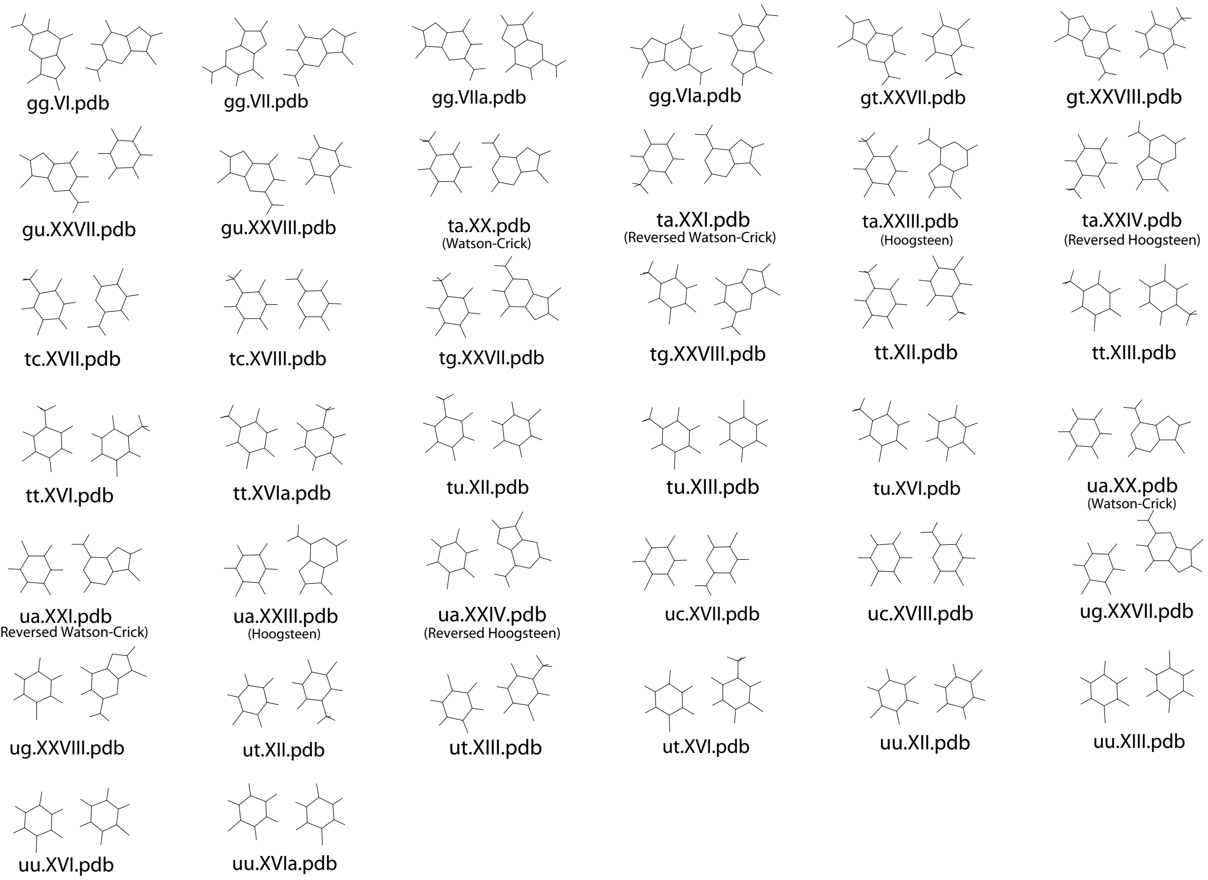
A typical use of these templates would be to set the bounds between two residues to some percentage of the idealized distance described by the template. In this case, the template would be the reference molecule (the second molecule passed to the function). A typical call might be:

```
useboundsfrom(b, m, "1:2,3:??,H?",
          getpdb( PATH + "gc.bdna.pdb" ), "::??,H?", 0.1 );
```

where PATH is $AMBERHOME/dat/dgdb/stacking/. This call sets the bounds of all the base atoms in residues 2 (GUA) and 3 (CYT) of strand 1 to be within 10% of the distances found in the template.

The basepair templates are named so that the first field of the template name is the one-character initials of the two individual residues and the next field is the Roman numeral corresponding to same bonding scheme described by Sanger, p. 120. *Note: since no specific sugar or backbone conformation is assumed in the templates, the non-base atoms should not be referenced.* The base atoms of the templates are show in figures 38.1 and 38.2.

The stacking templates are named in the same manner as the basepair templates. The first two letters of the template name are the one-character initials of the two residues involved in the stacking scheme (5' residue, then 3' residue) and the second field is the actual helical pattern (*note: a-rna represents the helical parameters of a'rna*). The stacking shemes can be found in the *$AMBERHOME/dat/dgdb/stacking* directory.



gg.VI.pdb gg.VII.pdb gg.VIIa.pdb gg.VIa.pdb gt.XXVII.pdb gt.XXVIII.pdb

gu.XXVII.pdb gu.XXVIII.pdb ta.XX.pdb (Watson-Crick) ta.XXI.pdb (Reversed Watson-Crick) ta.XXIII.pdb (Hoogsteen) ta.XXIV.pdb (Reversed Hoogsteen)

tc.XVII.pdb tc.XVIII.pdb tg.XXVII.pdb tg.XXVIII.pdb tt.XII.pdb tt.XIII.pdb

tt.XVI.pdb tt.XVIa.pdb tu.XII.pdb tu.XIII.pdb tu.XVI.pdb ua.XX.pdb (Watson-Crick)

ua.XXI.pdb (Reversed Watson-Crick) ua.XXIII.pdb (Hoogsteen) ua.XXIV.pdb (Reversed Hoogsteen) uc.XVII.pdb uc.XVIII.pdb ug.XXVII.pdb

ug.XXVIII.pdb ut.XII.pdb ut.XIII.pdb ut.XVI.pdb uu.XII.pdb uu.XIII.pdb

uu.XVI.pdb uu.XVIa.pdb

Figure 38.2.: *Basepair templates for use with* usebounds from(), *(gg-uu)*

38.4. Bounds databases

In addition to canonical templates, it is also possible to specify bounds information from a database of known molecular structures. This provides the option to use data obtained from actual structures, rather than from an idealized, canonical conformation.

The function setboundsfromdb() sets the bounds of all pairs of atoms between the two residues selected by aex1 and aex2 to a statistically averaged distance calculated from known structures plus or minus a multiple of the standard deviation. The statistical information is kept in database files. Currently, there are three types of database files - Those containing bounds information between Watson-Crick basepairs, those containing bounds information between helically stacked residues, and those containing intra-residue bounds information for residues in any conformation. The standard deviation is multiplied by the parameter *mul* and subtracted from the average distance to determine the lower bound and similarly added to the average distance to determine the upper bound of all base-base atom distances. Base-backbone bounds, that is, bounds between pairs of atoms in which one atom is a base atom and the other atom is a backbone atom, are set to be looser than base-base atoms. Specifically, the lower bound between a base-backbone atom pair is set to the smallest measured distance of all the structures considered in creating the database. Similarly, the upper bound between a base-backbone atom pair is set to the largest measured distance of all the structures considered. Base-base, and base-sugar bounds are set in a similar manner. This was done to avoid imposing false constraints on the atomic bounds, since Watson-Crick basepairing and stacking does not preclude any specific backbone and sugar conformation. setboundsfromdb() first searches the current directory for *dbase* before checking the default database location, $AMBERHOME/dat/dgdb.

Each entry in the database file has six fields: The atoms whose bounds are to be set, the number of separate structures sampled in constructing these statistics, the average distance between the two atoms, the standard deviation, the minimum measured distance, and the maximum measured distance. For example, the database bdna.basepair.db has the following sample entries:

```
A:C2-T:C1' 424 6.167 0.198 5.687 6.673
A:C2-T:C2 424 3.986 0.175 3.554 4.505
A:C2-T:C2' 424 7.255 0.304 5.967 7.944
A:C2-T:C3' 424 8.349 0.216 7.456 8.897
A:C2-T:C4 424 4.680 0.182 4.122 5.138
A:C2-T:C4' 424 8.222 0.248 7.493 8.800
A:C2-T:C5 424 5.924 0.168 5.414 6.413
A:C2-T:C5' 424 9.385 0.306 8.273 10.104
A:C2-T:C6 424 6.161 0.163 5.689 6.679
A:C2-T:C7 424 7.205 0.184 6.547 7.658
```

The first column identifies the atoms from the adenosine C2 atom to various thymidine atoms in a Watson-Crick basepair. The second column indicates that 424 structures were sampled in determining the next four columns: the average distance, the standard deviation, and the minimum and maximum distances.

The databases were constructing using the coordinates from all the known nucleic acid structures from the Nucleic Acid Database (NDB - http://www.ndbserver.ebi.ac.uk:5700/NDB/. If one wishes to remake the databases, the coordinates of all the NDB structures should be downloaded and kept in the $AMBERHOME/dat/coords directory. The databases are made by issuing the command $AMBERHOME/dat/dgdb/make_databases *dblist* where *dblist* is a list of nucleic acid types (i.e., bdna, arna, *etc.*). If one wants to add new structures to the structure repository at $AMBERHOME/dat/coords, it is necessary to make sure that the first two letters of the pdb file identify the nucleic acid type. That is, all bdna pdb files must begin with *bd*.

The nab functions used to create the databases are located in $AMBERHOME/dat/dgdb/functions. The stacking databases were constructed as follows: If two residues stacked 5' to 3' in a helix have fewer than ten inter-residue atom distances closer than 2.0 Å or larger than 9.0 Å, and if the normals between the base planes are less than 20.0o, the residues were considered stacked. The base plane is calculated as the normal to the N1-C4 and midpoint of the C2-N3 and N1-C4 vectors. The first atom expression given to setboundsfromdb() specifies the 5' residue and the second atom expression specifies the 3' residue. The source for this function is getstackdist.nab.

Similarly, the basepair databases were constructed by measuring the heavy atom distances of corresponding residues in a helix to check for hydrogen bonding. Specifically, if an A-U basepair has an N1-N3 distance of between 2.3 and 3.2 Å and a N6-O4 distance of between 2.3 and 3.3 Å, then the A-U basepair is considered a Waton-Crick basepair and is used in the database. A C-G basepair is considered Watson-Crick paired if the N3-N1 distance is between 2.3 and 3.3 Å, the N4-O6 distance is between 2.3 and 3.2 Å, and the O2-N2 distance is between 2.3 and 3.2 Å.

The nucleotide databases contain all the distance information between atoms in the same residue. No residues in the coordinates directory are excluded from this database. The intent was to allow the residues of this database to assume all possible conformations and ensure that a nucleotide residue would not be biased to a particular conformation.

For the basepair and stacking databases, setting the parameter *mul* to 1.0 results in lower bounds being set from the average database distance minus one standard deviation, and upper bounds as the average database distance plus one standard deviation, between base-base atoms. Base-backbone and base-sugar upper and lower bounds are set to the maximum and minimum measured database values, respectively. *Note, however, that a stacking multiple of 0.0 may not correspond to consistent bounds. A stacking multiple of 0.0 will probably have conflicting bounds information as the bounds information is derived from many different structures.*

The database types are named *nucleic_acid_type.database_type.*`db`, and can be found in the
$AMBERHOME/dat/dgdb directory.

39. NAB: Molecular mechanics and dynamics

The initial models created by rigid-body transformations or distance geometry are often in need of further refinement, and molecular mechanics and dynamics can often be useful here. nab has facilities to allow molecular mechanics and molecular dynamics calculations to be carried out. At present, this uses the AMBER program LEaP to set up the parameters and topology; the force field calculations and manipulations like minimization and dynamics are done by routines in the nab suite. A version of LEaP is included in the NAB distribution, and is accessed by the leap() discussed below. A later chapter gives a more detailed description.

39.1. Basic molecular mechanics routines

```
molecule getpdb_prm( string pdbfile, string leaprc, string leap_cmd2, int savef );
int readparm( molecule m, string parmfile );
int mme_init( molecule mol, string aexp, string aexp2, point xyz_ref[], file f );
int mm_options( string opts );
float mme( point xyz[], point grad[], int iter );
float mme_rattle( point xyz[], point grad[], int iter );
int conjgrad( float x[], int n, float fret, float func(), float rmsgrad,
        float dfpred, int maxiter );
int md( int n, int maxstep, point xyz[], point f[], float v[], float func );
int getxv( string filename, int natom, float start_time, float x[], float v[] );
int putxv( string filename, string title, int natom, float start_time,
        float x[], float v[] );
void mm_set_checkpoint( string filename );
```

The getpdb_prm() is a lot like getpdb() itself, except that it creates a molecule (and the associated force field parameters) that can be used in subsequent molecular mechanics calculations. It is often adequate to convert an input PDB file into a NAB molecule. (If this routine fails, you may be able to fix things up by editing your input pdb file, and/or by modifying the *leaprc* or *leap_cmd2* strings; if this doesn't work you will have to run tleap by hand, create a prmtop file, and use readparm() to input it.)

The leaprc string is passed to LEaP, and identifies which parameter and force field libraries to load. Sample leaprc files are in $AMBERHOME/dat/leap/cmd, and there is no default. The leap_cmd2 string is interpreted after the molecule has been read in to a unit called "X". Typically, leap_cmd2 would modify the molecule, say by adding or removing bonds, etc. The final parameter, savef will save the intermediate files if nonzero; otherwise, all intermediate files created will be removed. getpdb_prm() returns a molecule whose force field parameters are already populated, and hence is ready for further force-field manipulation.

readparm reads an AMBER parameter-topology file, created by tleap or with other AMBER programs, and sets up a data structure which we call a "parmstruct". This is part of the molecule, but is not directly accessible (yet) to nab programs. You would use this command as an alternative to getpdb_prm(). You need to be sure that the molecule used in the readparm() call has been created by calling getpdb() with a PDB file that has been created by tleap itself (i.e., that has exactly the Amber atoms in the correct order). As noted above, the readparm() routine is primarily intended for cases where getpdb_prm() fails (i.e., when you need to run *tleap* by hand).

setxyz_from_mol() copies the atomic coordinates of mol to the array xyz. setmol_from_xyz() replaces the atomic coordinates of mol with the contents of xyz. Both return the number of atoms copied with a 0 indicating an error occurred.

The getxv() and putxv() routines read and write non-periodic Amber-style restart files. Velocities are read if present.

The getxyz() and putxyz() routines are used in conjunction with the mm_set_checkpoint() routine to write checkpoint or restart files. The coordinates are written at higher precision than to an AMBER restart file, i.e., with sufficiently high precision to restart even a Newton-Raphson minimization where the error in coordinates may be on the order of10^{-12} . The checkpoint files are written at iteration intervals that are specified by the *nchk* or *nchk2* parameters to the mm_options() routine (see below). The checkpoint file names are determined by the filename string that is passed to mm_set_checkpoint(). If filename contains one or more %d format specifiers, then the file name will be a modification of filename wherein the leftmost %d of filename is replaced by the iteration count. If filename contains no %d format specifier, then the file name will be filename with the iteration count appended on the right.

The mme_init() function must be called after mm_options() and before calls to mme(). It sets up parameters for future force field evaluations, and takes as input an nab molecule. The string aexp is an atom expression that indicates which atoms are to be allowed to move in minimization or dynamics: atoms that do not match aexp will have their positions in the gradient vector set to zero. A NULL atom expression will allow all atoms to move. The second string, aexp2 identifies atoms whose positions are to be restrained to the positions in the array xyz_ref. The strength of this restraint will be given by the wcons variable set in mm_options(). A NULL value for aexp2 will cause all atoms to be constrained. The last parameter to mme_init() is a file pointer for the output trajectory file. This should be NULL if no output file is desired. NAB writes trajectories in the *binpos* format, which can be read by *ptraj*, and either analyzed, or converted to another format.

mm_options() is used to set parameters, and must be called before mme_init(); if you change options through a call to mm_options() without a subsequent call to mme_init() you may get incorrect calculations with no error messages. Beware. The opts string contains keyword/value pairs of the form keyword=value separated by white space or commas. Allowed values are shown in the following table.

keyword	*default*	*meaning*
ntpr	10	Frequency of printing of the energy and its components.
e_debug	0	If nonzero printout additional components of the energy.
gb_debug	0	If nonzero printout information about Born first derivatives.
gb2_debug	0	If nonzero printout information about Born second derivatives.
nchk	10000	Frequency of writing checkpoint file during first derivative calculation, i.e., in the mme() routine.
nchk2	10000	Frequency of writing checkpoint file during second derivative calculation, i.e., in the mme2() routine.
nsnb	25	Frequency at which the non-bonded list is updated.
nscm	0	If > 0, remove translational and rotational center-of-mass (COM) motion after every nscm steps. For Langevin dynamics (gamma_ln>0) without HCP (hcp=0), the position of the COM is reset to zero every nscm steps, but the velocities are not affected. With HCP (hcp>0) COM translation and rotation are also removed, with or without Langevin dynamics. It is strongly recommended that this option be used whenever HCP is used.
cut	8.0	Non-bonded cutoff, in angstroms. This parameter is ignored if hcp > 0.
wcons	0.0	Restraint weight for keeping atoms close to their positions in xyz_ref (see *mme_init*).
dim	3	Number of spatial dimensions; supported values are 3 and 4.
k4d	1.0	Force constant for squeezing out the fourth dimensional coordinate, if dim=4. If this is nonzero, a penalty function will be added to the bounds-violation energy, which is equal to 0.5 * *k4d* * *w* * *w*, where *w* is the value of the fourth dimensional coordinate.
dt	0.001	Time step, ps.

keyword	*default*	*meaning*
t	0.0	Initial time, ps.
rattle	0	If set to 1, bond lengths will be constrained to their equilibrium values, for dynamics; if set to 2, bonds to hydrogens will be constrained; default is not to include such constraints. Note: if you want to use rattle (effectively "shake") for minimization, you do not need to set this parameter; rather, pass the mme_rattle() function to *conjgrad()*.
tautp	999999.	Temperature coupling parameter, in ps. The time constant determines the strength of the weak-coupling ("Berendsen") temperature bath.[347] Set *tautp* to a very large value (e.g. 9999999.) in order to turn off coupling and revert to Newtonian dynamics. This variable only has an effect if *gamma_ln* remains at its default value of zero; if *gamma_ln* is not zero, Langevin dynamics is assumed, as discussed below.
gamma_ln	0.0	Collision frequency for Langevin dynamics, inps^{-1} . Values in the range 2-5ps^{-1} often give acceptable temperature control, while allowing transitions to take place.[356] Values near 50ps^{-1} correspond to the collision frequency for liquid water, and may be useful if rough physical time scales for motion are desired. The so-called BBK integrator is used here.[658]
temp0	300.0	Target temperature, K.
vlimit	20.0	Maximum absolute value of any component of the velocity vector.
ntpr_md	10	Printing frequency for dynamics information to stdout.
ntwx	0	Frequency for dumping coordinates to traj_file.
zerov	0	If nonzero, then the initial velocities will be set to zero.
tempi	0.0	If *zerov*=0 and *tempi*>0, then the initial velocities will be randomly chosen for this temperature. If both *zerov* and *tempi* are zero, the velocities passed into the md() function will be used as the initial velocities; this combination is useful to continue an existing trajectory.
genmass	10.0	The general mass to use for MD if individual masses are not read from a prmtop file; value in amu.
diel	C	Code for the dielectric model. "C" gives a dielectric constant of 1; "R" makes the dielectric constant equal to distance in angstroms; "RL" uses the sigmoidal function of Ramstein & Lavery, PNAS **85**, 7231 (1988); "RL94" is the same thing, but speeded up assuming one is using the Cornell *et al* force field; "R94" is a distance-dependent dielectric, again with speedups that assume the Cornell *et al.* force field.
dielc	1.0	This is the dielectric constant used for *non-GB* simulations. It is implemented in routine mme_init() by scaling all of the charges by sqrt(dielc). This means that you need to set this (if desired) in mm_options() before calling mme_init().

keyword	*default*	*meaning*
gb	0	If set to 0 then GB is off. Setting gb=1 turns on the Hawkins, Cramer, Truhlar (HCT) form of pairwise generalized Born model for solvation. See ref [154] for details of the implementation; this is equivalent to the *igb=1* option in *sander* and *pmemd*. Set diel to "C" if you use this option. Setting gb=2 turns on the Onufriev, Bashford, Case (OBC) variant of GB,[133, 138] with α=0.8, β=0.0 and γ=2.909. This is equivalent to the *igb=2* option in *sander* and *pmemd*. Setting gb=5 just changes the values of α, β and γ to 1.0, 0.8, and 4.85, respectively, corresponding to the *igb=5* option in *sander*. Setting gb=7 turns on the GB Neck variant of GB,[156] corresponding to the *igb=7* option in *sander* and *pmemd*. Setting gb=8 turns on the updated GB Neck variant of GB, corresponding to the *igb=8* option in *sander* and *pmemd*.
rgbmax	999.0	A maximum value for considering pairs of atoms to contribute to the calculation of the effective Born radii. The default value means that there is effectively no cutoff. Calculations will be sped up by using smaller values, say around 15. Å or so. This parameter is ignored if hcp > 0.
gbsa	0	If set to 1, add a surface-area dependent energy equal to surften*SASA, where surften is discussed below, and SASA is an approximate surface area term. NAB uses the "LCPO" approximation developed by Weiser, Shenkin, and Still.[126]
surften	0.005	Surface tension (see *gbsa*, above) in kcal/mol/Å^2.
epsext	78.5	Exterior dielectric for generalized Born; interior dielectric is always 1.
kappa	0.0	Inverse of the Debye-Hueckel length, if gb is turned on, in Å^{-1}. This parameter is related to the ionic strength as $\kappa = [8\pi\beta I/\varepsilon]^{1/2}$, where I is the ionic strength (same as the salt concentration for a 1-1 salt). For T=298.15 and ε=78.5, $\kappa = (0.10806 I)^{1/2}$, where I is in [M].
ipb	0	Switch to compute electrostatic solvation free energy. If set to 0 then PBSA is off. This is equivalent to the ipb option in *pbsa*. Possible values: **0**, **1**, **2**, and **4**. See PBSA chapter for more information.
inp	2	Option to select different methods to compute non-polar solvation free energy. This is equivalent to the inp option in *pbsa*. Possible values: **0**, **1**, and **2**. See PBSA chapter for more information.
epsin	1.0	Sets the dielectric constant of the solute region. The solute region is defined to be the solvent excluded volume.
epsout	80.0	Sets the implicit solvent dielectric constant. The solvent region is defined to be the space not occupied the solute region. Thus, only two dielectric regions are allowed in the current release.
smoothopt	1	Instructs PB how to set up dielectric values for finite-difference grid edges that are located across the solute/solvent dielectric boundary.
istrng	0.0	Sets the ionic strength (in mM) for the PB equation.
radiopt	1	Option to set up atomic radii. This is equivalent to the radiopt option in *pbsa*. Possible values: **0**, and **1**. See PBSA chapter for more information.

keyword	*default*	*meaning*
dprob	1.4	Solvent probe radius for molecular surface used to define the dielectric boundary between solute and solvent. If set 0.0, it would be later assigned to the value of sprob.
iprob	2.0	Mobile ion probe radius for ion accessible surface used to define the Stern layer.
npbopt	0	Option to select the linear or the full nonlinear PB equation. = **0** Linear PB equation is solved. = **1** Nonlinear PB equation is solved.
solvopt	1	Option to select iterative solvers. This is equivalent to the solvopt option in *pbsa*. Possible values: **1**, **2**, **3**, **4**, **5**, and **6**. See PBSA chapter for more information.
accept	0.001	Sets the iteration convergence criterion (relative to the initial residue).
maxitn	100	Sets the maximum number of iterations for the finite difference solvers, default to 100.
fillratio	2.0	The ratio between the longest dimension of the rectangular finite-difference grid and that of the solute.
space	0.5	Sets the grid spacing for the finite difference solver.
nfocus	2	Set how many successive FD calculations will be used to perform an electrostatic focussing calculation on a molecule. Possible values: **1** and **2**.
fscale	8	Set the ratio between the coarse and fine grid spacings in an electrostatic focussing calculation.
bcopt	5	Boundary condition options. This is equivalent to the bcopt option in *pbsa*. Possible values: **1**, **5**, **6**, and **10**. See PBSA chapter for more information.
eneopt	2	Option to compute total electrostatic energy and forces. This is equivalent to the eneopt option in *pbsa*. Possible values: **1**, and **2**. See PBSA chapter for more information.
dbfopt	n/a	This keyword is phased out in this release.
frcopt	0	Option to compute and output electrostatic forces to a file named force.dat in the working directory. This is equivalent to the frcopt option in *pbsa*. Possible values: **0**, **1**, **2**, and **3**. See PBSA chapter for more information.
cutnb	0.0	Atom-based cutoff distance for van der Waals interactions, and pairwise Coulombic interactions when ENEOPT = 2. When ENEOPT = 1, this is the cutoff distance used for van der Waals interactions only.
sprob	0.557	Solvent probe radius for solvent accessible surface area (SASA) used to compute the dispersion term.
npbverb	0	This turns on verbose mode in PB when set to 1.
arcres	0.25	gives the resolution (in the unit of Å) of dots used to represent solvent accessible arcs.
maxarcdot	1500	1500 actually means automatically determine number of arc dots required for solvent accessible surface, might grow too large to fit machines with less available memory. Please assign it to 4000~7000 and see if it fits into your computers.
npbgrid	1	How many step do pbsa wait to re-calculate the geometry in a simulation, npbgrid = 1 is required to do trajectory evaluation. npbgrid is recommended to be 100 if “conjgrad” is used.

keyword	*default*	*meaning*
irism	0	Use 3D-RISM. **= 0** Off. **= 1** On.
xvvfile	n/a	.xvv file which describes bulk solvent properties. Required for 3D-RISM calculations. Produced by rism1d.
guvfile	n/a	Root name for solute-solvent 3D pair distribution function, $G^{\mathrm{UV}}(\mathbf{R})$. This will produce one file for each solvent atom type for each frame requested.
huvfile	n/a	Rootname for solute-solvent 3D total correlation function, $H^{\mathrm{UV}}(\mathbf{R})$. This will produce one file for each solvent atom type for each frame requested.
cuvfile	n/a	Rootname for solute-solvent 3D total correlation function, $C^{\mathrm{UV}}(\mathbf{R})$. This will produce one file for each solvent atom type for each frame requested.
quvfile	n/a	Rootname for solvent 3D charge density distribution [e/Å]. This will produce one file with contributions from each solvent atom type for each frame requested.
chgdist	n/a	Rootname for solvent 3D charge distribution [e]. This will produce one file with contributions from each solvent atom type for each frame requested.
uuvfile	n/a	Rootname for solute-solvent 3D potential energy, $U^{\mathrm{UV}}(\mathbf{R})$. This will produce one file for each solvent atom type for each frame requested.
asympfile	n/a	Rootname for solute-solvent 3D long range real-space asymptotics for C and H. This will produce one file for C and H for each frame requested.
volfmt	dx	Output format for volumetric data. **= dx** DX format. **= xyzv** XYZV format.
closure	KH	Comma separate list of closure approximations. **= HNC** Hyper-netted chain equation (HNC). **= KH** Kovalenko-Hirata (KH). **= PSEn** Partial series expansion of order n where “n” is a positive integer. If more than one closure is provided, the 3D-RISM solver will use the closures in order to obtain a solution for the last closure in the list when no previous solutions are available. The solution for the last closure in the list is used for all output.
closureorder	1	(Deprecated) Order for PSE-n closure if closure is specified as “PSE” or “PSEN” (no integers).
solvcut	buffer	Cut-off distance for solvent-solute potential and force calculations. `solvcut` must be explicitly set if buffer<0. For minimization it is recommended to not use a cut-off (e.g. solvcut=`9999`).

keyword	*default*	*meaning*
buffer	14	Minimum distance in Å between the solute and the edge of the solvent box. **< 0** Use fixed box size (`ng3` and `solvbox`). **>= 0** Buffer distance.
grdspc	0.5	Linear grid spacing in x-, y- and z-dimensions [Å]. May be specified as single number if all dimensions have the same value. E.g., ‘grdspc=0.5’ is equivalent to ‘grdspc=0.5,0.5,0.5’.
ng	n/a	Sets the number of grid points for a fixed size solvation box. May be specified as single integer if all dimensions have the same value. E.g., ‘ng=64’ is equivalent to ‘ng=64,64,64’.
solvbox	n/a	Sets the size in Å of the fixed size solvation box. May be specified as single number if all dimensions have the same value. E.g., ‘solvbox=32.0’ is equivalent to ‘solvbox=32.0,32.0,32.0’.
tolerance	1e-5	A list of maximum residual values for solution convergence. When used in combination with a list of closures it is possible to define different tolerances for each of the closures. This can be useful for difficult to converge calculations (see §7.3.1). For the sake of efficiency, it is best to use as high a tolerance as possible for all but the last closure. For minimization a tolerance of 1e-11 or lower is recommended. Three formats of list are possible. one tolerance All closures but the last use a tolerance of 1. The last tolerance in the list is used by the last closure. In practice this, is the most efficient. two tolerances All closures but the last use the first tolerance in the list. The last tolerance in the list is used by the last closure. *n* tolerances Tolerances from the list are assigned to the closure list in order.
mdiis_del	0.7	“Step size” in MDIIS.
mdiis_nvec	5	Number of vectors used by the MDIIS method. Higher values for this parameter can greatly increase memory requirements but may also accelerate convergence.
mdiis_restart	10	If the current residual is `mdiis_restart` times larger than the smallest residual in memory, then the MDIIS procedure is restarted using the lowest residual solution stored in memory. Increasing this number can sometimes help convergence.
mdiis_method	2	Specify implementation of the MDIIS routine. **= 0** Original reference implementation. **= 1** BLAS optimized. **= 2** BLAS and memory optimized.

keyword	*default*	*meaning*
maxstep	10000	Maximum number of iterations allowed to converge on a solution.
npropagate	5	Number of previous solutions propagated forward to create an initial guess for this solute atom configuration. **= 0** Do not use any previous solutions **= 1..5** Values greater than 0 but less than 4 or 5 will use less system memory but may introduce artifacts to the solution (e.g., energy drift).
centering	1	Controls how the solute is centered/re-centered in the solvent box. (See Subsection 7.5.1.) **= -4** Center-of-geometry with grid-point rounding. Center on first step only. **= -3** Center-of-mass with grid-point rounding. Center on first step only. **= -2** Center-of-geometry. Center on first step only. **= -1** Center-of-mass. Center on first step only. **= 0** No centering. Dangerous. **= 1** Center-of-mass. Center on every step. Recommended for molecular dynamics. **= 2** Center-of-geometry. Center on every step. Recommended for minimization. **= 3** Center-of-mass with grid-point rounding. **= 4** Center-of-geometry with grid-point rounding.
zerofrc	1	Redistribute solvent forces across the solute such that the net solvation force on the solute is zero. **= 0** Unmodified forces. **= 1** Zero net force.
apply_rism_force	1	Calculate and use solvation forces from 3D-RISM. Not calculating these forces can save computation time and is useful for trajectory post-processing. **= 0** Do not calculate forces. **= 1** Calculate forces.

keyword	*default*	*meaning*
ntwrism	0	Indicates that solvent density grid should be written to file every `ntwrism` iterations. **= 0** No files written. **>= 1** Output every ntwrism time steps.
ntprism	0	Indicates that 3D-RISM thermodynamic output should be written to file every `ntprism` iterations. **= 0** No files written. **>= 1** Output every ntwrism time steps.
polarDecomp	0	Decompose the solvation free energy into polar and non-polar contributions. This is only useful if ntprism$\neq 0$ and adds about 80% to the total calculation time. **= 0** No decomposition. **= 1** Decomposition is performed.
entropicDecomp	0	Decomposes solvation free energy into energy and entropy components. Also performs temperature derivatives of other calculated quantities. Note that this typically requires 80% more computation time and requires a .xvv file version 1.000 or higher (see §7.1.3 and 7.3). **= 0** No entropic decomposition. **= 1** Entropic decomposition.
gf	0	Compute the Gaussian fluctuation excess chemical potential functional (see §7.1.2).
pcpluscorrection	0	Compute the PC+/3D-RISM excess chemical potential functional (see §7.2.4).
uccoeff	0,0,0,0	Compute the UC excess chemical potential functional with the provided coefficients (see §7.2.4). *a* and *b* are the coefficients for the original UC functional, though using the closure excess chemical potential functional. *a1* and *b1* are optional and provide temperature dependence to the correction (UCT in [228]).
verbose	0	Indicates level of diagnostic detail about the calculation written to the log file. **= 0** No output. **= 1** Print the number of iterations required to converge. **= 2** Print details for each iteration and information about what FCE is doing every progress iterations.
progress	1	Display progress of the 3D-RISM solution every `progress` iterations. 0 indicates this information will not be displayed. Only used if `verbose` > 1.

keyword	*default*	*meaning*
static_arrays	1	If set to 1, do not allocate dynamic arrays for each call to the mme() and mme2() functions. The default value of 1 reduces computation time by avoiding array allocation.
blocksize	8	The granularity with which loop iterations are assigned to OpenMP threads or MPI processes. For MPI, a blocksize as small as 1 results in better load balancing during parallel execution. For OpenMP, blocksize should not be smaller than the number of floating-point numbers that fit into one cache line in order to avoid performance degradation through 'false sharing'. For ScaLAPACK, the optimum blocksize is not know, although a value of 1 is probably too small.
hcp	0	Use the GB-HCP model: **= 0** No GB-HCP. **= 1** 1-charge approximation. **= 2** 2-charge approximation. **= 4** 2-charge based on optimal point charge approximation (recommended for GB-HCP). See Section 39.6 for detailed instructions on using the GB-HCP. It is strongly recommended that the NSCM option above be used whenever GB-HCP is used.
dhcp	0.25	Adjusts the separation between the charges used to approximate uncharged components for hcp=4. dhcp is empirically determined so that the RMS error in force, compared to GB without further approximation, is minimized. Our testing on various structures suggests that the optimal value for dhcp can be found within the range of 0.1 and 0.4. See Section 39.6 for details.
hcp_h1	15	GB-HCP level 1 threshold distance. The recommended level 1 threshold distance for amino acids is 15A. For structures with nucleic acids the recommended level 1 threshold distance is 21A.
hcp_h2	50	GB-HCP level 2 threshold distance. The recommended level 2 threshold distance for proteins is 50A. For structures with nucleic acids the recommended level 2 threshold distance is 90A.
hcp_h3	150	GB-HCP level 3 threshold distance. The recommended level 3 threshold distance for amino acids is 150A. For structures with nucleic acids the recommended level 1 threshold distance is 169A.

The mme() function takes a coordinate set and returns the energy in the function value and the gradient of the energy in grad. The input parameter *iter* is used to control printing (see the *ntpr* variable) and non-bonded updates (see *nsnb*). The mme_rattle() function has the same interface, but constrains the bond lengths and returns a corrected gradient. If you want to minimize with constrained bond lengths, pass *mme_rattle* and not *mme* to the *conjgrad* routine.

The conjgrad() function will carry out conjugate gradient minimization of the function func that depends upon n parameters, whose initial values are in the x array. The function func must be of the form func(x[], g[], iter), where x contains the input values, and the function value is returned through the function call, and its gradient with respect to x through the g array. The iteration number is passed through iter, which func can use for whatever purpose it wants; a typical use would just be to determine when to print results. The input parameter dfpred is the expected drop in the function value on the first iteration; generally only a rough estimate is needed. The minimization will proceed until maxiter steps have been performed, or until the root-mean-square of the components of the gradient is less than rmsgrad. The value of the function at the end of the minimization is returned in the variable fret. conjgrad can return a variety of exit codes:

Return codes for conjgrad routine	
>0	minimization converged; gives number of final iteration
-1	bad line search; probably an error in the relation of the function to its gradient (perhaps from round-off if you push too hard on the minimization).
-2	search direction was uphill
-3	exceeded the maximum number of iterations
-4	could not further reduce function value

Finally, the md function will run maxstep steps of molecular dynamics, using func as the force field (this would typically be set to a function like mme.) The number of dynamical variables is given as input parameter n: this would be 3 times the number of atoms for ordinary cases, but might be different for other force fields or functions. The arrays x[], f[] and v[] hold the coordinates, gradient of the potential, and velocities, respectively, and are updated as the simulation progresses. The method of temperature regulation (if any) is specified by the variables *tautp* and *gamma_ln* that are set in *mm_options()*.

Note: In versions of NAB up to 4.5.2, there was an additional input variable to md() called *minv* that reserved space for the inverse of the masses of the particles; this has now been removed. This change is not backwards compatible: you must modify existing NAB scripts that call md() to remove this variable.

39.2. NetCDF read/write routines

NAB has several routines for reading/writing Amber NetCDF trajectory and restart files. All of the routines except netcdfGetNextFrame() return a 1 on error, 0 on success. The netcdfGetNextFrame() routine returns 0 on error, 1 on success to make it easier to use in loops. For an example of how to use NetCDF files in NAB see the NAB script in **’$AMBERHOME/AmberTools/test/nab/tnetcdf.nab’**.

39.2.1. struct AmberNetcdf

An AmberNetcdf struct must be used to interface with the netcdf commands in NAB (except netcdfWriteRestart()). It contains many fields, but the following are the ones commonly needed by users:

temp0 Temperature of current frame (if temperature is present).

restartTime Simulation time if NetCDF restart.

isNCrestart 0 if trajectory, 1 if restart.

ncframe Number of frames in the file.

currentFrame Current frame number.

ncatom Number of atoms.

ncatom3 Number of coordinates (ncatom * 3).

velocityVID If not -1, velocity information is present.

TempVID If not -1, temperature information is present.

In order to use it, you must include nab_netcdf.h and declare it as a struct, e.g.:

```
#include "nab_netcdf.h"
struct AmberNetcdf NC;
```

39.2.2. netcdfClose

```
int netcdfClose(struct AmberNetcdf NC)
```

Close NetCDF file associated with **NC**.

39.2.3. netcdfCreate

```
int netcdfCreate(struct AmberNetcdf NC, string filename, int natom, int isBox)
```

NC `AmberNetcdf struct to set up.`

filename `Name of file to create.`

natom `Number of atoms in file.`

isBox `0 = No box coordinates, 1 = Has box coordinates.`

Create NetCDF trajectory file and associate with struct **NC**. For writing NetCDF restarts, use netcdfWriteRestart().

39.2.4. netcdfDebug

```
int netcdfDebug(struct AmberNetcdf NC)
```

Print debug information for NetCDF file associated with **NC**.

39.2.5. netcdfGetFrame

```
int netcdfGetFrame(struct AmberNetcdf NC, int set, float X[], float box[])
```

NC `AmberNetcdf struct, previously set up and opened.`

set `Frame number to read.`

X `Array to store coordinates (dimension NC.ncatom3).`

box `Array of dimension 6 to store box coordinates if present (X Y Z ALPHA BETA GAMMA); can be NULL.`

Get coordinates at frame **set** (starting from 0).

39.2.6. netcdfGetNextFrame

```
int netcdfGetNextFrame(struct AmberNetcdf NC, float X[], float box[])
```

NC `AmberNetcdf struct, previously set up and opened.`

X `Array to store coordinates (dimension NC.ncatom3).`

box `Array of size 6 to store box coordinates if present (X Y Z ALPHA BETA GAMMA); can be NULL.`

Get the coordinates at frame **NC.currentFrame** and increment **NC.currentFrame** by one. Unlike the other netcdf routines, this returns 1 on success and 0 on error to make it easy to use in loops.

39.2.7. netcdfGetVelocity

```
int netcdfGetVelocity(struct AmberNetcdf NC, int set, float V[])
```

NC `AmberNetcdf struct, previously set up and opened.`

set `Frame number to read.`

V `Array to store velocities (dimension NC.ncatom3).`

Get velocities at frame **set** (starting from 0).

39.2.8. netcdfInfo

```
int netcdfInfo(struct AmberNetcdf NC)
```

Print information for **NC**, including file type, presence of velocity/box/temperature info, and number of atoms, coordinates, and frames present.

39.2.9. netcdfLoad

```
int netcdfLoad(struct AmberNetcdf NC, string filename)
```

NC `AmberNetcdf struct to set up.`

filename `Name of NetCDF file to load.`

Load NetCDF file filename and set up the AmberNetcdf structure **NC** for reading. The file type is automatically detected.

39.2.10. netcdfWriteFrame

```
int netcdfWriteFrame(struct AmberNetcdf NC, int set, float X[], float box[])
```

NC `AmberNetcdf struct, previously set up and opened.`

set `Frame number to write.`

X `Array of coordinates to write (dimension NC.ncatom3).`

box `Array of size 6 of box coordinates to write (X Y Z ALPHA BETA GAMMA); can be NULL.`

Write to NetCDF trajectory at frame **set** (starting from 0). NOTE: This routine is for writing NetCDF trajectories only; to write NetCDF restarts use netcdfWriteRestart().

39.2.11. netcdfWriteNextFrame

```
int netcdfWriteNextFrame(struct AmberNetcdf NC, float X[], float box[])
```

NC `AmberNetcdf struct, previously set up and opened.`

X `Array of coordinates to write (dimension NC.ncatom3).`

box `Array of size 6 of box coordinates to write (X Y Z ALPHA BETA GAMMA); can be NULL.`

Write coordinates to frame **NC.currentFrame** and increment **NC.currentFrame** by one. NOTE: This routine is for writing NetCDF trajectories only; to write NetCDF restarts use netcdfWriteRestart().

39.2.12. netcdfWriteRestart

```
int netcdfWriteRestart(string filename, int natom, float X[], float V[],
                       float box[], float time, float temperature)
```

filename Name of NetCDF restart file to create.

natom Number of atoms in netcdf restart file.

X Array of coordinates to write (dimension natom*3).

V Array of velocities to write (dimension natom*3); can be NULL.

box Array of size 6 of box coordinates to write (X Y Z ALPHA BETA GAMMA); can be NULL.

time Restart time in ps.

temperature Restart temperature; if < 0 no temperature will be written.

39.3. Typical calling sequences

The following segment shows some ways in which these routines can be put together to do some molecular mechanics and dynamics:

```
1  // carry out molecular mechanics minimization and some simple dynamics
2  molecule m, mi;
3  int  ier;
4  float  m_xyz[ dynamic ], f_xyz[ dynamic ], v[ dynamic ];
5  float  dgrad, fret, dummy[2];
6
7  mi = bdna( "gcgc" );
8  putpdb( "temp.pdb", mi );
9  m = getpdb_prm( "temp.pdb", "leaprc.nucleic.OL15", "", 0 );
10
11 allocate m_xyz[ 3*m.natoms ];  allocate f_xyz[ 3*m.natoms ];
12 allocate v[ 3*m.natoms ];
13 setxyz_from_mol( m, NULL, m_xyz );
14
15 mm_options( "cut=25.0, ntpr=10, nsnb=999, gamma_ln=5.0" );
16 mme_init( m, NULL, "::ZZZ", dummy, NULL );
17 fret = mme( m_xyz, f_xyz, 1 );
18 printf( "Initial energy is %8.3f\n", fret );
19
20 dgrad = 0.1;
21 ier = conjgrad( m_xyz, 3*m.natoms, fret, mme, dgrad, 10.0, 100 );
22 setmol_from_xyz( m, NULL, m_xyz );
23 putpdb( "gcgc.min.pdb", m );
24
25 mm_options( "tautp=0.4, temp0=100.0, ntpr_md=10, tempi=50." );
26 md( 3*m.natoms, 1000, m_xyz, f_xyz, v, mme );
27 setmol_from_xyz( m, NULL, m_xyz );
28 putpdb( "gcgc.md.pdb", m );
```

Line 7 creates an nab molecule; any nab creation method could be used here. Then a temporary pdb file is created, and this is used to generate a NAB molecule that can be used for force-field calculations (line 9). Lines 11-13 allocate some memory, and fill the coordinate array with the molecular position. Lines 15-17 initialize the force field routine, and call it once to get the initial energy. The atom expression "::ZZZ" will match no atoms, so that there will be no restraints on the atoms; hence the fourth argument to mme_init can just be a place-holder, since there are no reference positions for this example. Minimization takes place at line 21, which will call mme repeatedly, and which also arranges for its own printout of results. Finally, in lines 25-28, a short (1000-step) molecular dynamics run is made. Note the the initialization routine mme_init *must* be called before calling the evaluation routines mme or md.

Elaboration of the the above scheme is generally straightforward. For example, a simulated annealing run in which the target temperature is slowly reduced to zero could be written as successive calls to mm_options (setting the temp0 parameter) and md (to run a certain number of steps with the new target temperature.) Note also that routines other than mme could be sent to conjgrad and md: any routine that takes the same three arguments and returns a float function value could be used. In particular, the routines db_viol (to get violations of distance bounds from a bounds matrix) or mme4 (to compute molecular mechanics energies in four spatial dimensions) could be used here. Or, you can write your own nab routine to do this as well. For some examples, see the *gbrna*, *gbrna_long* and *rattle_md* programs in the $AMBERHOME/AmberTools/test/nab directory.

39.4. Second derivatives and normal modes

Russ Brown has contributed new codes that compute analytically the second derivatives of the Amber functions, including the generalized Born terms. This capability resides in the three functions described here.

int newton(float x[], int n, float fret, float func1(), float func2(), float rms,
float nradd, int maxiter);
float nmode(float x[], int n, float func(), int eigp, int ntrun, float eta, float hrmax, int ioseen);

These routines construct and manipulate a Hessian (second derivative matrix), allowing one (for now) to carry out Newton-Raphson minimization and normal mode calculations. The mme2() routine takes as input a *3*natom* vector of coordinates x[], and returns a gradient vector g[], a Hessian matrix, stored columnwise in a *3*natom x 3*natom* vector h[], and the masses of the system, in a vector m[] of length *natom*. The iteration variable iter is just used to control printing. At present, these routines only work for *gb* = 0 or 1.

Users cannot call mme2() directly, but will pass this as an argument to one of the next two routines.

The newton() routine takes a input coordinates x[] and a size parameter n (must be set to *3*natom*). It performs Newton-Raphson optimization until the root-mean-square of the gradient vector is less than rms, or until maxiter steps have been taken. For now, the input function func1() must be mme() and func2() must be mme2(). The value nradd will be added to the diagonal of the Hessian before the step equations are solved; this is generally set to zero, but can be set something else under particular circumstances, which we do not discuss here.[659]

Generally, you only want to try Newton-Raphson minimization (which can be very expensive) after you have optimized structures with conjgrad() to an rms gradient of 10 -3 or so. In most cases, it should only take a small number of iterations then to go down to an rms gradient of about 10 -12 or so, which is somewhere near the precision limit.

Once a good minimum has been found, you can use the nmode() function to compute normal/Langevin modes and thermochemical parameters. The first three arguments are the same as for newton(), the next two integers give the number of eigenvectors to compute and the type of run, respectively. The last three arguments (only used for Langevin modes) are the viscosity in centipoise, the value for the hydrodynamic radius, and the type of hydrodynamic interactions. Several techniques are available for diagonalizing the Hessian depending on the number of modes required and the amount of memory available.

In all cases the modes are written to an Amber-compatible "vecs" file for normal modes or "lmodevecs" file for Langevin modes. There are currently no nab routines that use this format. The Langevin modes will also generate an output file called "lmode" that can be read by the Amber module *lmanal*.

ntrun	**0:** The dsyev routine is used to diagonalize the Hessian **1:** The dsyevd routine is used to diagonalize the Hessian **2:** The ARPACK package (shift invert technique) is used to obtain a small number of eigenvalues **3:** The Langevin modes are computed with the viscosity and hydrodynamic radius provided
hrmax	Hydrodynamic radius for the atom with largest area exposed to solvent. If a file named "expfile" is provided then the relative exposed areas are read from this file. If "expfile" is not present all atoms

are assigned a hydrodynamic radius of hrmax or 0.2 for the hydrogen atoms. The "expfile" can be generated with the ms (molecular surface) program.

ioseen	**0:** Stokes Law is used for the hydrodynamic interaction
	1: Oseen interaction included
	2: Rotne-Prager correction included

Here is a typical calling sequence:

```
molecule m;
float x[4000], fret;

m = getpdb_prm( "mymolecule.pdb", "leaprc.protein.ff14SB", "", 0 );
mm_options( "cut=999., ntpr=50, nsnb=99999, diel=C, gb=1, dielc=1.0" );
mme_init( m, NULL, "::Z", x, NULL);
setxyz_from_mol( m, NULL, x );

//  conjugate gradient minimization
conjgrad(x, 3*m.natoms, fret, mme, 0.1, 0.001, 2000 );

//  Newton-Raphson minimization\fP
mm_options( "ntpr=1" );
newton( x, 3*m.natoms, fret, mme, mme2, 0.00000001, 0.0, 6 );

// get the normal modes:
nmode( x, 3*m.natoms, mme2, 0, 0, 0.0, 0.0, 0);
```

39.5. Low-MODe (LMOD) optimization methods

István Kolossváry has contributed new functions, which implement the LMOD methods for minimization, conformational searching, and flexible docking.[405–408] The centerpiece of LMOD is a conformational search algorithm based on eigenvector following of low-frequency vibrational modes. It has been applied to a spectrum of computational chemistry domains including protein loop optimization and flexible active site docking. The search method is implemented without explicit computation of a Hessian matrix and utilizes the Arnoldi package (ARPACK, *http://www.caam.rice.edu/software/ARPACK/*) for computing the low-frequency modes. LMOD optimization can be thought of as an advanced minimization method. LMOD can not only energy minimize a molecular structure in the local sense, but can generate a series of very low energy conformations. The LMOD capability resides in a single, top-level calling function *lmod()*, which uses fast local minimization techniques, collectively termed XMIN that can also be accessed directly through the function *xmin()*.

New for AmberTools17: there are now **four "real-life" examples** of carrying out LMOD searches: look in *$AMBERHOME/AmberTools/examples/nab/lmod_**. Each directory has a README file that give more information.

39.5.1. LMOD conformational searching

The LMOD conformational search procedure is based on gentle, but very effective structural perturbations applied to molecular systems in order to explore their conformational space. LMOD perturbations are derived from low- frequency vibrational modes representing large-amplitude, concerted atomic movements. Unlike essential dynamics where such low modes are derived from long molecular dynamics simulations, LMOD calculates the modes directly and utilizes them to improve Monte Carlo sampling.

LMOD has been developed primarily for macromolecules, with its main focus on protein loop optimization. However, it can be applied to any kind of molecular systems, including complexes and flexible docking where it has found widespread use. The LMOD procedure starts with an initial molecular model, which is energy minimized.

The minimized structure is then subjected to an ARPACK calculation to find a user-specified number of low-mode eigenvectors of the Hessian matrix. The Hessian matrix is never computed; ARPACK makes only implicit reference to it through its product with a series of vectors. Hv, where v is an arbitrary unit vector, is calculated via a finite-difference formula as follows,

$$Hv = [\nabla(x_{min} + h) - \nabla(x_{min})]/h \tag{39.1}$$

where x_{min} is the coordinate vector at the energy minimized conformation and h denotes machine precision. The computational cost of Eq. 1 requires a single gradient calculation at the energy minimum point and one additional gradient calculation for each new vector. Note that $\bigtriangledown x$ is never 0, because minimization is stopped at a finite gradient RMS, which is typically set to 0.1-1.0 kcal/mol-Å in most calculations.

The low-mode eigenvectors of the Hessian matrix are stored and can be re-used throughout the LMOD search. Note that although ARPACK is very fast in relative terms, a single ARPACK calculation may take up to a few hours on an absolute CPU time scale with a large protein structure. Therefore, it would be impractical to recalculate the low-mode eigenvectors for each new structure. Visual inspection of the low-frequency vibrational modes of different, randomly generated conformations of protein molecules showed very similar, collective motions clearly suggesting that low-modes of one particular conformation were transferable to other conformations for LMOD use. This important finding implies that the time limiting factor in LMOD optimization, even for relatively small molecules, is energy minimization, not the eigenvector calculation. This is the reason for employing XMIN for local minimization instead of NAB's standard minimization techniques.

39.5.2. LMOD Procedure

Given the energy-minimized structure of an initial protein model, protein- ligand complex, or any other molecular system and its low-mode Hessian eigenvectors, LMOD proceeds as follows. For each of the first n low-modes repeat steps 1-3 until convergence:

1. Perturb the energy-minimized starting structure by moving along the ith (i =1-n) Hessian eigenvector in either of the two opposite directions to a certain distance. The 3N-dimensional (N is equal to the number of atoms) travel distance along the eigenvector is scaled to move the fastest moving atom of the selected mode in 3-dimensional space to a randomly chosen distance between a user-specified minimum and maximum value.
   Note: A single LMOD move inherently involves excessive bond stretching and bond angle bending in Cartesian space. Therefore the primarily torsional trajectory drawn by the low-modes of vibration on the PES is severely contaminated by this naive, linear approximation and, therefore, the actual Cartesian LMOD trajectory often misses its target by climbing walls rather than crossing over into neighboring valleys at not too high altitudes. The current implementation of LMOD employs a so-called ZIG-ZAG algorithm, which consists of a series of alternating short LMOD moves along the low-mode eigenvector (ZIG) followed by a few steps of minimization (ZAG), which has been found to relax excessive stretches and bends more than reversing the torsional move. Therefore, it is expected that such a ZIG- ZAG trajectory will eventually be dominated by concerted torsional movements and will carry the molecule over the energy barrier in a way that is not too different from finding a saddle point and crossing over into the next valley like passing through a mountain pass.
   Barrier crossing check: The LMOD algorithm checks barrier crossing by evaluating the following criterion: IF the current endpoint of the zigzag trajectory is lower than the energy of the starting structure, OR, the endpoint is at least lower than it was in the previous ZIG-ZAG iteration step AND the molecule has also moved farther away from the starting structure in terms of all-atom superposition RMS than at the previous position THEN it is assumed that the LMOD ZIG-ZAG trajectory has crossed an energy barrier.

2. Energy-minimize the perturbed structure at the endpoint of the ZIG- ZAG trajectory.

3. Save the new minimum-energy structure and return to step 1. Note that LMOD saves only low-energy structures within a user-specified energy window above the then current global minimum of the ongoing search.

Parameter list for xmin()		
keyword	*default*	*meaning*
func	N/A	The name of the function that computes the function value and gradient of the objective function to be minimized. *func()* must have the following argument list: `float func( float x[], float g[], int i)` where `x[]` is the vector of the iterate, `g[]` is the gradient and `i` is currently ignored except when `func` = `mme` where `i` is handled internally.
natm	N/A	Number of atoms. **NOTE:** if `func` is other than `mme`, `natm` is used to pass the total number of variables of the objective function to be minimized. However, `natm` retains its original meaning in case `func` is a user-defined energy function for 3-dimensional (molecular) structure optimization. Make sure that the meaning of `natm` is compatible with the setting of `mol_struct_opt` below.
x[]	N/A	Coordinate vector. User has to allocate memory in calling program and fill x[] with initial coordinates using, e.g., the setxyz_from_mol function (see sample program below). Array size = 3*natm.
g[]	N/A	Gradient vector. User has to allocate memory in calling program. Array size = 3*natm.
ene	N/A	On output, ene stores the minimized energy.
grms_out	N/A	On output, grms_out stores the gradient RMS achieved by XMIN.

Table 39.2.: *Arguments for xmin().*

After exploring the modes of a single structure, LMOD goes on to the next starting structure, which is selected from the set of previously found low- energy structures. The selection is based on either the Metropolis criterion, or simply the than lowest energy structure is used. LMOD terminates when the user-defined number of steps has been completed or when the user-defined number of low-energy conformations has been collected.

Note that for flexible docking calculations LMOD applies explicit translations and rotations of the ligand(s) on top of the low-mode perturbations.

39.5.3. XMIN

```
float xmin( float func(), int natm, float x[], float g[],
               float ene, float grms_out, struct xmod_opt xo);
```

At a glance: The *xmin()* function minimizes the energy of a molecular structure with initial coordinates given in the x[] array. On output, *xmin()* returns the minimized energy as the function value and the coordinates in x[] will be updated to the minimum-energy conformation. The arguments to xmin() are described in Table 39.2; the parameters in the xmin_opt structure are described in Table 39.3; these should be preceded by "`xo.`", since they are members of an *xmod_opt* struct with that name; see the sample program below to see how this works.

There are three types of minimizers that can be used, specified by the *method* parameter:

method

1: PRCG Polak-Ribiere conjugate gradient method, similar to the *conjgrad()* function [409].

2: L-BFGS Limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton algorithm [410]. L-BFGS is 2-3 times faster than PRCG mainly, because it requires significantly fewer line search steps than PRCG.

3: lbfgs-TNCG L-BFGS preconditioned truncated Newton conjugate gradient algorithm [409, 411]. Sophisticated technique that can minimize molecular structures to lower energy and gradient than PRCG and L-BFGS and requires an order of magnitude fewer minimization steps, but L-BFGS can sometimes be faster in terms of total CPU time.

Parameter list for xmin_opt		
keyword	*default*	*meaning*
mol_struct_opt	1	*1*= 3-dimensional molecular structure optimization. Any other value means general function optimization.
maxiter	1000	Maximum number of iteration steps allowed for XMIN. A value of zero means single point energy calculation, no minimization.
grms_tol	0.05	Gradient RMS threshold below which XMIN should minimize the input structure.
method	3	Minimization algorithm. See text for description.
numdiff	1	Finite difference method used in TNCG for approximating the product of the Hessian matrix and some vector in the conjugate gradient iteration (the same approximation is used in LMOD, see Eq. 39.1 in section 39.5.1). 1= Forward difference. 2=Central difference.
m_lbfgs	3	Size of the L-BFGS memory used in either L-BFGS minimization or L-BFGS preconditioning for TNCG. The value zero turns off preconditioning. It usually makes little sense to set the value >10.
print_level	0	Amount of debugging printout. 0= No output. 1= Minimization details. 2= Minimization (including conjugate gradient iteration in case of TNCG) and line search details.
iter	N/A	Output parameter. The total number of iteration steps completed by XMIN.
xmin_time	N/A	Output parameter. CPU time in seconds used by XMIN.
ls_method	2	*1*= modified Armijo [660](not recommended, primarily used for testing). *2*= Wolfe (after J. J. More' and D. J. Thuente).
ls_maxiter	20	Maximum number of line search steps per single minimization step.
ls_maxatmov	0.5	Maximum (co-ordinate) movement per degree of freedom allowed in line search, range > *0*.
beta_armijo	0.5	Armijo beta parameter, range (0, 1). *Only change it if you know what you are doing.*
c_armijo	0.4	Armijo c parameter, range (0, 0.5). *Only change it if you know what you are doing.*
mu_armijo	1.0	Armijo mu parameter, range [0, 2). *Only change it if you know what you are doing.*
ftol_wolfe	0.0001	Wolfe ftol parameter, range (0, 0.5). *Only change it if you know what you are doing.*
gtol_wolfe	0.9	Wolfe gtol parameter, range (ftol_wolfe, 1). *Only change it if you know what you are doing.*
ls_iter	N/A	Output parameter. The total number of line search steps completed by XMIN.
error_flag	N/A	Output parameter. A nonzero value indicates an error. In case of an error XMIN will always print a descriptive error message.

Table 39.3.: *Options for xmin_opt.*

4: Debugging option; printing analytical and numerical derivatives for comparison. Almost all failures with *xmin* can be attributed to inaccurate analytical derivatives, e.g., when SCF hasn't converged with a quantum based Hamiltonian.

NOTE: The xmin routine can be utilized for minimizing arbitrary, user-defined objective functions. The function must be defined in a user NAB program or in any other user library that is linked in. The name of the function is passed to *xmin()* via the *func* argument.

39.5.4. Sample XMIN program

The following sample program, which is based on the test program txmin.nab, reads a molecular structure from a PDB file, minimizes it, and saves the minimized structure in another PDB file.

```
//   XMIN reverse communication external minimization package.
//   Written by Istvan Kolossvary.

#include "xmin_opt.h"

// M A I N  P R O G R A M  to carry out XMIN minimization on a molecule:

struct xmin_opt xo;

molecule mol;
int natm;
float xyz[ dynamic ], grad[ dynamic ];
float energy, grms;
point dummy;

    xmin_opt_init( xo );  //  set up defaults (shown here)

    // xo.mol_struct_opt = 1;
    // xo.maxiter        = 1000;
    // xo.grms_tol       = 0.05;
    // xo.method         = 3;
    // xo.numdiff        = 1;
    // xo.m_lbfgs        = 3;
    //    xo.ls_method   = 2;
    //    xo.ls_maxiter  = 20;
    //    xo.maxatmov    = 0.5;
    //    xo.beta_armijo = 0.5;
    //    xo.c_armijo    = 0.4;
    //    xo.mu_armijo   = 1.0;
    //    xo.ftol_wolfe  = 0.0001;
    //    xo.gtol_wolfe  = 0.9;
    // xo.print_level    = 0;

    xo.maxiter     = 10;  //  non-defaults are here
    xo.grms_tol    = 0.001;
    xo.method      = 3;
    xo.ls_maxatmov = 0.15;
    xo.print_level = 2;

    mol = getpdb( "gbrna.pdb" );
    readparm( mol, "gbrna.prmtop" );
    natm = mol.natoms;
    allocate xyz[ 3*natm ]; allocate grad[ 3*natm ];
    setxyz_from_mol( mol, NULL, xyz );

    mm_options( "ntpr=1, gb=1, kappa=0.10395, rgbmax=99., cut=99.0, diel=C ");
    mme_init( mol, NULL, "::ZZZ", dummy, NULL );

    energy = mme( xyz, grad, 0 );
    energy = xmin( mme, natm, xyz, grad, energy, grms, xo );

// E N D  M A I N
```

The corresponding screen output should look similar to this. Note that this is fairly technical, debugging information; normally print_level is set to zero.

```
Reading parm file (gbrna.prmtop)
title:
PDB 5DNB, Dickerson decamer
old prmtop format => using old algorithm for GB parms
      mm_options:  ntpr=99
      mm_options:  gb=1
      mm_options:  kappa=0.10395
      mm_options:  rgbmax=99.
      mm_options:  cut=99.0
      mm_options:  diel=C
    iter    Total    bad         vdW     elect.     cons.   genBorn   frms

ff:   0  -4107.50    906.22   -192.79   -137.96      0.00  -4682.97 1.93e+01


----------------------------------------------------------------

 MIN:                         It=    0  E=   -4107.50 ( 19.289)
  CG:   It=    3 (  0.310)  :-)
  LS: step= 0.94735  it= 1  info= 1
 MIN:                         It=    1  E=   -4423.34 (  5.719)
  CG:   It=    4 (  0.499)  :-)
  LS: step= 0.91413  it= 1  info= 1
 MIN:                         It=    2  E=   -4499.43 (  2.674)
  CG:   It=    9 (  0.498)  :-)
  LS: step= 0.86829  it= 1  info= 1
 MIN:                         It=    3  E=   -4531.20 (  1.543)
  CG:   It=    8 (  0.499)  :-)
  LS: step= 0.95556  it= 1  info= 1
 MIN:                         It=    4  E=   -4547.59 (  1.111)
  CG:   It=    9 (  0.491)  :-)
  LS: step= 0.77247  it= 1  info= 1
 MIN:                         It=    5  E=   -4556.35 (  1.068)
  CG:   It=    8 (  0.361)  :-)
  LS: step= 0.75150  it= 1  info= 1
 MIN:                         It=    6  E=   -4562.95 (  1.042)
  CG:   It=    8 (  0.273)  :-)
  LS: step= 0.79565  it= 1  info= 1
 MIN:                         It=    7  E=   -4568.59 (  0.997)
  CG:   It=    5 (  0.401)  :-)
  LS: step= 0.86051  it= 1  info= 1
 MIN:                         It=    8  E=   -4572.93 (  0.786)
  CG:   It=    4 (  0.335)  :-)
  LS: step= 0.88096  it= 1  info= 1
 MIN:                         It=    9  E=   -4575.25 (  0.551)
  CG:   It=   64 (  0.475)  :-)
  LS: step= 0.95860  it= 1  info= 1
 MIN:                          It=   10  E=   -4579.19 (  0.515)
----------------------------------------------------------------

 FIN:             :-)                    E=   -4579.19 (  0.515)
```

The first few lines are typical NAB output from mm_init() and mme(). The output below the horizontal line comes from XMIN. The MIN/CG/LS blocks contain the following pieces of information. The MIN: line shows the current iteration count, energy and gradient RMS (in parentheses). The CG: line shows the CG iteration count and the residual in parentheses. The happy face :-) means convergence whereas :-(indicates that CG iteration encountered negative curvature and had to abort. The latter situation is not a serious problem, minimization can continue. This is just a safeguard against uphill moves. The LS: line shows line search information. "step" is the relative step with respect to the initial guess of the line search step. "it" tells the number of line search steps taken and "info" is an error code. "info" = 1 means that line searching converged with respect to sufficient decrease and curvature criteria whereas a non- zero value indicates an error condition. Again, an error in line searching doesn't mean that minimization necessarily failed, it just cannot proceed any further because of some numerical dead end. The FIN: line shows the final result with a happy face :-) if either the grms_tol criterion has been met or when the number of iteration steps reached the maxiter value.

39.5.5. LMOD

```
float lmod( int natm, float x[], float g[], float ene, float conflib[],
       float lmod_traj[], int lig_start[], int lig_end[], int lig_cent[],
       float tr_min[], float tr_max[], float rot_min[], float rot_max[],
       struct xmin_opt, struct xmin_opt, struct lmod_opt);
```

At a glance: The *lmod()* function is similar to *xmin()* in that it optimizes the energy of a molecular structure with initial coordinates given in the x[] array. However, the optimization goes beyond local minimization, it is a sophisticated conformational search procedure. On output, *lmod()* returns the global minimum energy of the LMOD conformational search as the function value and the coordinates in x[] will be updated to the global minimum-energy conformation. Moreover, a set of the best low-energy conformations is also returned in the array conflib[]. Coordinates, energy, and gradient are in NAB units. The parameters are given in the table below; items above the line are passed as parameters; the rest of the parameters are all preceded by "`lo.`", because they are members of an *lmod_opt* struct with that name; see the sample program below to see how this works.

Also note that *xmin()*'s *xmin_opt* struct is passed to *lmod()* as well. *lmod()* changes the default values of some of the "`xo.`" parameters via the call to *lmod_opt_int()* relative to a call to *xmin_opt_init()*, which means that in a more complex NAB program with multiple calls to *xmin()* and *lmod()*; make sure to always initialize and set user parameters for each and every XMIN and LMOD search via, respectively calling *xmin_opt_init()* and *lmod_opt_init()* just before the calls to *xmin()* and *lmod()*.

keyword	*default*	*meaning*
natm		Number of atoms.
x[]		Coordinate vector. User has to allocate memory in calling program and fill x[] with initial coordinates using, e.g., the setxyz_from_mol function (see sample program below). Array size = 3*natm.
g[]		Gradient vector. User has to allocate memory in calling program. Array size = 3*natm.
ene		On output, ene stores the global minimum energy.
conflib[]		User allocated storage array where LMOD stores low-energy conformations. Array size = 3*natm*nconf.
lmod_traj[]		User allocated storage array where LMOD stores snapshots of the pseudo trajectory drawn by LMOD on the potential energy surface. Array size = 3*natom * (nconf + 1).

keyword	*default*	*meaning*
lig_start[]	N/A	The serial number(s) of the first/last atom(s) of the ligand(s). The number(s) should correspond to the numbering in the NAB input files. Note that the ligand(s) can be anywhere in the atom list, however, a single ligand must have continuous numbering between the corresponding lig_start and lig_end values. The arrays should be allocated in the calling program. Array size = nlig, but in case nlig=0 there is no need for allocating memory.
lig_end[]	N/A	See above.
lig_cent[]	N/A	Similar array in all respects to lig_start/end, but the serial number(s) define the center of rotation. The value zero means that the center of rotation will be the geometric center of gravity of the ligand.
tr_min[]	N/A	The range of random translation/rotation applied to individual ligand(s). Rotation is carried out about the origin defined by the corresponding lig_cent value(s). The angle is given in +/- degrees and the distance in angstroms. The particular angles and distances are randomly chosen from their respective ranges. The arrays should be allocated in the calling program. Array size = *nlig*, but in case *nlig=0* there is no need to allocate memory.
tr_max[]		See tr_min[], above.
rot_min[]		See tr_min[], above.
rot_max[]		See tr_min[], above.
niter	10	The number of LMOD iterations. Note that a single LMOD iteration involves a number of different computations (see section 39.5.2.). A value of zero results in a single local minimization; like a call to xmin.
nmod	5	The total number of low-frequency modes computed by LMOD every time such computation is requested.
minim_grms	0.1	The gradient RMS convergence criterion of structure minimization.
kmod	3	The definite number of randomly selected low-modes used to drive LMOD moves at each LMOD iteration step.
nrotran_dof	6	The number of rotational and translational degrees of freedom. This is related to the number of frozen or tethered atoms in the system: 0 atoms dof=6, 1 atom dof=3, 2 atoms dof=1, >=3 atoms dof=0. Default is 6, no frozen or tethered atoms. See section 39.5.7, note (5).
nconf	10	The maximum number of low-energy conformations stored in conflib[]. Note that the calling program is responsible for allocating memory for conflib[].
energy_window	50.0	The energy window for conformation storage; the energy of a stored structure will be in the interval [global_min, global_min + energy_window].
eig_recalc	5	The frequency, measured in LMOD iterations, of the recalculation of eigenvectors.
ndim_arnoldi	0	The dimension of the ARPACK Arnoldi factorization. The default, zero, specifies the whole space, that is, three times the number of atoms. See note below.

keyword	*default*	*meaning*
lmod_restart	10	The frequency, in LMOD iterations, of updating the conflib storage, that is, discarding structures outside the energy window, and restarting LMOD with a randomly chosen structure from the low-energy pool defined by n_best_struct below. A value >maxiter will prevent LMOD from doing any restarts.
n_best_struct	10	Number of the lowest-energy structures found so far at a particular LMOD restart point. The structure to be used for the restart will be chosen randomly from this pool. n_best_struct = 1 allows the user to explore the neighborhood of the then current global minimum.
mc_option	1	The Monte Carlo method. 1= Metropolis Monte Carlo (see rtemp below). 2= "Total_Quench", which means that the LMOD trajectory always proceeds towards the lowest lying neighbor of a particular energy well found after exhaustive search along all of the randomly selected kmod low-modes. 3= "Quick_Quench", which means that the LMOD trajectory proceeds towards the first neighbor found, which is lower in energy than the current point on the path, without exploring the remaining modes.
rtemp	1.5	The value of RT in NAB energy units. This is utilized in the Metropolis criterion.
lmod_step_size_min	2.0	The minimum length of a single LMOD ZIG move in Å. See section 39.5.2.
lmod_step_size_max	5.0	The maximum length of a single LMOD ZIG move in Å. See section 39.5.2.
nof_lmod_steps	0	The number of LMOD ZIG-ZAG moves. The default, zero, means that the number of ZIG-ZAG moves is not pre-defined, instead LMOD will attempt to cross the barrier in as many ZIG-ZAG moves as it is necessary. The criterion of crossing an energy barrier is stated above in section 39.5.2. nof_lmod_steps > 0 means that multiple barriers may be crossed and LMOD can carry the molecule to a large distance on the potential energy surface without severely distorting the geometry.
lmod_relax_grms	1.0	The gradient RMS convergence criterion of structure relaxation, see ZAG move in section 39.5.2.
nlig	0	Number of ligands considered for flexible docking. The default, zero, means no docking.
apply_rigdock	2	The frequency, measured in LMOD iterations, of the application of rigid-body rotational and translational motions to the ligand(s). At each apply_rigdock-th LMOD iteration nof_pose_to-try rotations and translations are applied to the ligand(s).
nof_poses_to_try	10	The number of rigid-body rotational and translational motions applied to the ligand(s). Such applications occur at each apply_rigdock-th LMOD iteration. In case nof_pose_to_try > 1, it is always the lowest energy pose that is kept, all other poses are discarded.
random_seed	314159	The seed of the random number generator. A value of zero requests hardware seeding based on the system clock.

keyword	*default*	*meaning*
print_level	0	Amount of debugging printout. 0= No output. 1= Basic output. 2= Detailed output. 3= Copious debugging output including ARPACK details.
lmod_time	N/A	CPU time in seconds used by LMOD itself.
aux_time	N/A	CPU time in seconds used by auxiliary routines.
error_flag	N/A	A nonzero value indicates an error. In case of an error LMOD will always print a descriptive error message.

Notes on the *ndim_arnoldi* parameter: Basically, the ARPACK package used for the eigenvector calculations solves multiple "small" eigenvalue problems instead of a single "large" problem, which is the diagonalization of the three times the number of atoms by three times the number of atoms Hessian matrix. This parameter is the user specified dimension of the "small" problem. The allowed range is nmod + 1 <= ndim_arnoldi <= 3*natm. The default means that the "small" problem and the "large" problem are identical. This is the preferred, i.e., fastest, calculation for small to medium size systems, because ARPACK is guaranteed to converge in a single iteration. The ARPACK calculation scales with three times the number of atoms times the Arnoldi dimension squared and, therefore, for larger molecules there is an optimal ndim_arnoldi much less than three times the number of atoms that converges much faster in multiple iterations (possibly thousands or tens of thousands of iterations). The key to good performance is to select ndim_arnoldi such that all the ARPACK storage fits in memory. For proteins, ndim_arnoldi =1000 is generally a good value, but often a very small ~50-100 Arnoldi dimension provides the fastest net computational cost with very many iterations.

39.5.6. Sample LMOD program

The following sample program, which is based on the test program tlmod.nab, reads a molecular structure from a PDB file, runs a short LMOD search, and saves the low-energy conformations in PDB files.

```
//   LMOD reverse communication external minimization package.
//   Written by Istvan Kolossvary.

#include "xmin_opt.h"
#include "lmod_opt.h"

// M A I N  P R O G R A M  to carry out LMOD simulation on a molecule/complex:

struct xmin_opt xo;
struct lmod_opt lo;

molecule mol;
int   natm;
float energy;
int lig_start[ dynamic ], lig_end[ dynamic ], lig_cent[ dynamic ];
float xyz[ dynamic ], grad[ dynamic ], conflib[ dynamic ], lmod_trajectory[ dynamic ];
float tr_min[ dynamic ], tr_max[ dynamic ], rot_min[ dynamic ], rot_max[ dynamic ];
float glob_min_energy;
point dummy;

    lmod_opt_init( lo, xo );   //  set up defaults

    lo.niter           = 3;     //  non-default options are here
    lo.mc_option       = 2;
    lo.nof_lmod_steps  = 5;
    lo.random_seed     = 99;
    lo.print_level     = 2;

    xo.ls_maxatmov    = 0.15;

    mol = getpdb( "trpcage.pdb" );
    readparm( mol, "trpcage.top" );
    natm = mol.natoms;

    allocate xyz[ 3*natm ]; allocate grad[ 3*natm ];
    allocate conflib[ lo.nconf * 3*natm ];
    allocate lmod_trajectory[ (lo.niter+1) * 3*natm ];
    setxyz_from_mol( mol, NULL, xyz );

    mm_options( "ntpr=5000, gb=0, cut=999.0, nsnb=9999, diel=R ");
    mme_init( mol, NULL, "::ZZZ", dummy, NULL );

    mme( xyz, grad, 1 );
    glob_min_energy = lmod( natm, xyz, grad, energy,
          conflib, lmod_trajectory, lig_start, lig_end, lig_cent,
          tr_min, tr_max, rot_min, rot_max, xo, lo );

    printf( "\nGlob. min. E         = %12.3lf kcal/mol\n", glob_min_energy );


// E N D  M A I N
```

The corresponding screen output should look similar to this.

```
Reading parm file (trpcage.top)
title:

      mm_options:  ntpr=5000
      mm_options:  gb=0
      mm_options:  cut=999.0
      mm_options:  nsnb=9999
      mm_options:  diel=R
________________________________________________________________
                  Low-Mode Simulation
---------------------------------------------------------------------------
     1    E =     -118.117 ( 0.054)  Rg =    5.440
  1  / 6  E =     -89.2057 ( 0.090)  Rg =    2.625  rmsd=  8.240  p= 0.0000
  1  / 8  E =      -51.682 ( 0.097)  Rg =    5.399  rmsd=  8.217  p= 0.0000
  3  /12  E =     -120.978 ( 0.091)  Rg =    3.410  rmsd=  7.248  p= 1.0000
  3  /10  E =     -106.292 ( 0.099)  Rg =    5.916  rmsd=  4.829  p= 0.0004
  4  / 6  E =     -106.788 ( 0.095)  Rg =    4.802  rmsd=  3.391  p= 0.0005
  4  / 3  E =     -111.501 ( 0.097)  Rg =    5.238  rmsd=  2.553  p= 0.0121
---------------------------------------------------------------------------
     2    E =     -120.978 ( 0.091)  Rg =    3.410
  1  / 4  E =     -137.867 ( 0.097)  Rg =    2.842  rmsd=  5.581  p= 1.0000
  1  / 9  E =     -130.025 ( 0.100)  Rg =    4.282  rmsd=  5.342  p= 1.0000
  4  / 3  E =     -123.559 ( 0.089)  Rg =    3.451  rmsd=  1.285  p= 1.0000
  4  / 4  E =     -107.253 ( 0.095)  Rg =    3.437  rmsd=  2.680  p= 0.0001
  5  / 5  E =     -113.119 ( 0.096)  Rg =    3.136  rmsd=  2.074  p= 0.0053
  5  / 4  E =       -134.1 ( 0.091)  Rg =    3.141  rmsd=  2.820  p= 1.0000
---------------------------------------------------------------------------
     3    E =     -130.025 ( 0.100)  Rg =    4.282
  1  / 8  E =     -150.556 ( 0.093)  Rg =    3.347  rmsd=  5.287  p= 1.0000
  1  / 4  E =     -123.738 ( 0.079)  Rg =    4.218  rmsd=  1.487  p= 0.0151
  2  / 8  E =     -118.254 ( 0.095)  Rg =    3.093  rmsd=  5.296  p= 0.0004
  2  / 7  E =     -115.027 ( 0.090)  Rg =    4.871  rmsd=  4.234  p= 0.0000
  4  / 7  E =     -128.905  ( 0.099)  Rg =    4.171  rmsd=  2.113  p= 0.4739
  4  /11  E =      -133.85  ( 0.099)  Rg =    3.290  rmsd=  4.464  p= 1.0000
-----------------------------------------------------------
 Full list:
      1  E =     -150.556 / 1  Rg =    3.347
      2  E =     -137.867 / 1  Rg =    2.842
      3  E =       -134.1 / 1  Rg =    3.141
      4  E =      -133.85 / 1  Rg =    3.290
      5  E =     -130.025 / 1  Rg =    4.282
      6  E =     -128.905 / 1  Rg =    4.171
      7  E =     -123.738 / 1  Rg =    4.218
      8  E =     -123.559 / 1  Rg =    3.451
      9  E =     -120.978 / 1  Rg =    3.410
     10  E =     -118.254 / 1  Rg =    3.093

Glob. min. E         =     -150.556 kcal/mol
```

The first few lines come from *mm_init()* and *mme()*. The screen output below the horizontal line originates from LMOD. Each LMOD-iteration is represented by a multi-line block of data numbered in the upper left corner by the iteration count. Within each block, the first line displays the energy and, in parentheses, the gradient RMS as well as the radius of gyration (assigning unit mass to each atom), of the current structure along the LMOD pseudo simulation-path. The successive lines within the block provide information about the LMOD ZIG-ZAG moves (see section 39.5.2). The number of lines is equal to 2 times kmod (2x3 in this example). Each selected mode is explored in both directions, shown in two separate lines. The leftmost number is the serial number of the mode (randomly selected from the set of nmod modes) and the number after the slash character gives the number of ZIG-ZAG moves taken. This is followed by, respectively, the minimized energy and gradient RMS, the radius of gyration, the RMSD distance from the base structure, and the Boltzmann probability with respect to the energy of the base structure and rtemp, of the minimized structure at the end of the ZIG-ZAG path. Note that exploring the same mode along both directions can result in two quite different structures. Also note that the number of ZIG-ZAG moves required to cross the energy barrier (see section 39.5.2) in different directions can vary quite a bit, too. Occasionally, an exclamation mark next to the energy (!E = ...) denotes a structure that could not be fully minimized.

After finishing all the computation within a block, the corresponding LMOD step is completed by selecting one of the ZIG-ZAG endpoint structures as the base structure of the next LMOD iteration. The selection is based on the *mc_option* and the Boltzmann probability. The LMOD pseudo simulation-path is defined by the series of these *mc_option*-selected structures and it is stored in *lmod_traj[]*. Note that the sample program saves these structures in a multi- PDB disk file called *lmod_trajectory.pdb*. The final section of the screen output lists the nconf lowest energy structures found during the LMOD search. Note that some of the lowest energy structures are not necessarily included in the *lmod_traj[]* list, as it depends on the *mc_option* selection. The list displays the energy, the number of times a particular conformation was found (increasing numbers are somewhat indicative of a more complete search), and the radius of gyration. The glob. min. energy is printed from the sample NAB program, not from LMOD. The sample program in *$AMBERHOME/AmberTools/examples/nab/lmod_dock* shows how one could write the top ten low-energy structures in separate, numbered PDB files.

As a final note, it is instructive to be aware of a simple safeguard that LMOD applies . A copy of the *conflib[]* array is saved periodically in a binary disk file called *conflib.dat*. Since LMOD searches might run for a long time, in case of a crash low-energy structures can be recovered from this file. The format of *conflib.dat* is as follows. Each conformation is represented by 3 numbers (double energy, double radius of gyration, and int number of times found), followed by the double (x, y, z) coordinates of the atoms.

39.5.7. Tricks of the trade of running LMOD searches

1. The AMBER atom types HO, HW, and ho all have zero van der Waals parameters in all of the AMBER (and some other) force fields. Corresponding Aij and Bij coefficients in the PRMTOP file are set to zero. This means there is no repulsive wall to prevent two oppositely charged atoms, one being of type HO,

HW or ho, to fuse as a result of the ever decreasing electrostatic energy as they come closer and closer to each other. This potential problem is rarely manifest in molecular dynamics simulations, but it presents a nuisance when running LMOD searches. The problem is local minimization, especially "aggressive" TNCG minimization (XMIN xo.method=3) that can easily result in atom fusion. Therefore, before running an LMOD simulation, the PRMTOP file (let's call it prmtop.in) must be processed by running the script "lmodprmtop prmtop.in prmtop.out". This script will replace all the repulsive Aij coefficients set to zero in prmtop.in with a high value of 1e03 in prmtop.out in order to re-create the van der Waals wall. It is understood that this procedure is parameter fudging; however, note that the primary goal of using LMOD is the quick generation of approximate, low-energy structures that can be further refined by high-accuracy MD.

2. LMOD requires that the potential energy surface is continuous everywhere to a great degree. Therefore, always use a distance dependent dielectric constant in mm_options when running searches in vacuo, or use GB solvation (note that GB calculations will be slow), and always apply a large cut-off. It does make sense to run quick and dirty LMOD searches in vacuo to generate low-energy starting structures for MD runs. Note that the most likely symptom of discontinuities causing a problem is when your NAB program utilizing LMOD is grabbing CPU time, but the LMOD search does not seem to progress. This is the result of NaN's that often can be seen when print_level is set to > 0.

3. LMOD is NOT INTENDED to be used with explicit water models and periodic boundary conditions. Although explicit-water solvation representation is not recommended, LMOD docking can be readily used with crystallographic water molecules as ligands.

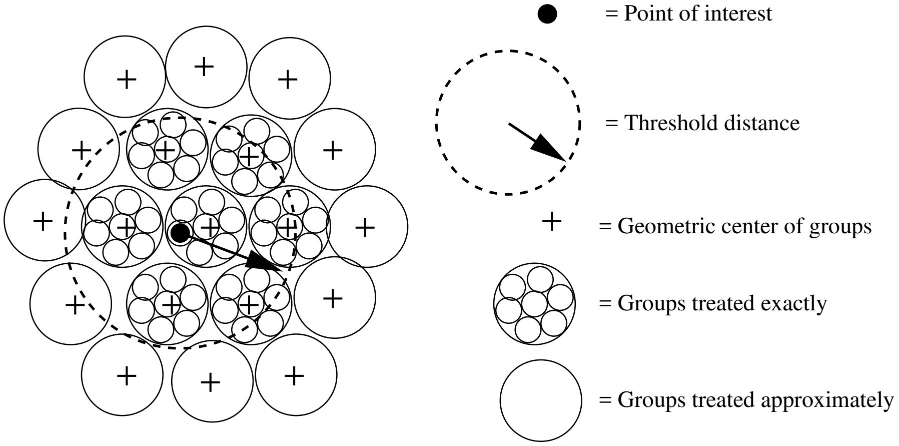
4. Conformations in the conflib and lmod_trajectory files can have very different orientations. One trick to keep them in a common orientation is to restrain the position of, e.g., a single benzene ring. This will ensure that the molecule cannot be translated or rotated as a whole. However, when applying this trick you should set nrotran_dof = 0.

5. A subset of the atoms of a molecular system can be frozen or tethered/restrained in NAB by two different methods. Atoms can either be frozen by using the first atom expression argument in *mme_init()* or restrained by using the second atom expression argument and the reference coordinate array in *mme_init()* along with the *wcons* option in mm_options. LMOD searches, especially docking calculations can be run much faster if parts of the molecular system can be frozen, because the effective degrees of freedom is determined by the size of the flexible part of the system. Application of frozen atoms means that a much smaller number of moving atoms are moving in the fixed, external potential of the frozen atoms. The tethered atom model is expected to give similar results to the frozen atom model, but note that the number of degrees of freedom and, therefore, the computational cost of a tethered calculation is comparable to that of a fully unrestrained system. However, the eigenvector calculations are likely to converge faster with the tethered systems.

39.6. The Generalized Born with Hierarchical Charge Partitioning (GB-HCP)

GB-HCP (and its latest version, GB-HCPO[661]) is a multi-scale, yet fully atomistic, approach to perform MD simulations based on the generalized Born model, mainly intended for large and very large structures. For example, it was used to refine a 1.1M atom structure of 30nm chromatin fiber[661]. Compared to the reference GB model without further approximations, GB-HCP can deliver up to 3 orders of magnitude speedup, depending on structure size. In contrast to cutoff GB that completely ignores the effect of long range electrostatic interactions beyond a certain distance, which can lead to serious artifacts under many circumstances such as for highly charged systems, GB-HCP takes into account the long range electrostatic interactions by using N log N Hierarchical Charge Partitioning (HCP) approximation [662, 663]. Based on this method, structures are partitioned into multiple hierarchical levels of components using the natural organization of the biomolecular structures - atoms, groups, chains, and complexes. The charge distribution for each of these components is approximated by 1 (hcp=1) or 2 (hcp=2

Figure 39.1.: *The HCP threshold distance. For the level 1 approximation shown here, groups within the threshold distance are treated exactly using atomic charges, while groups beyond the threshold distance are approximated by a small number of charges, e.g. 1 charge for hcp=1 shown here.*



and hcp=4) charges. Setting hcp=4 (strongly recommended) uses GB-HCPO, which takes advantage of the Optimal Point Charge Approximation approach for placing the approximate point charges[87]: two point charges are placed so that the three lowest order multipole moments of the reference charge distribution are optimally reproduced. The approximate charges are then used for computing electrostatic interactions with distant components while the full set of atomic charges are used for nearby components (Figure 40.1). The HCP can be used for generalized Born (gb=1-8) simulations, for gas phase (dielec=C) and distant dependent dielectric (dielec=R/RL), with or without Langevin dynamics (gamma_ln>0).

The usage of the new feature (hcp=4) requires that the separation between the two charges used to approximate the uncharged components is specified by dhcp. The value of dhcp is empirically adjusted so that the RMS error in force, compared to the GB without further approximation, is minimized. Our testing on a various set of structures suggests that dhcp=0.25 is optimal for many systems. However, if further accuracy is desired for specific systems, the value for dhcp can be further optimized within the range of 0.1 and 0.4 following the steps below. To find the optimal value for hcp, one time step simulation for the starting configuration of the structure can be performed using the GB model without approximation (hcp=0), and with e_debug=1setting, that automatically prints out the forces on each atom into a text file called reference.frc. Rename reference.frc to exact.frc. Then, run one step of the starting configuration of the structure using the GB-HCP (hcp=4) by setting the dhcp parameter within the range of 0.1 and 0.4 in increments of 0.05. The reference.frc file produced for each value of dhcp can be compared to the exact.frc to compute the RMS error in force. The following command line computes the RMS error:

```
paste exact.frc reference.frc | awk '{x+=($9-$20)^2+($10-$21)^2+($11-$22)^2}END{print sqrt(x/NR)}'
```

The optimal value for dhcp is the one that results in minimum RMS error in the force.

39.6.1. Level 1 HCP approximation

The HCP option can now be used with one level of approximation (groups) using the NAB molecular dynamics scripts described in Section 39.3 above. No additional manipulation of the input structure files is required for one level of approximation. For an example see AmberTools/examples/hcp/2trx.nab. The level 1 approximation is recommended for single domain and small (< 10,000 atoms) multi-domain structures. Speedups of 2x-10x can be realized using the level 1 approximation, depending on structure size.

39.6.2. Level 2 and 3 HCP approximation

For larger multi-domain structures higher levels of approximations (chains and complexes) can be used to achieve up to 3 orders of magnitude speedups, depending on structure size. The following additional steps are required to include information about these higher level components in the prmtop file. For an example see AmberTools/examples/hcp/1kx5.nab. A fully working example (including the MD run scripts) of a 3 level partitioning of a giant structure, one million atom chromatin fiber, can be found at http://people.cs.vt.edu/onufriev/software.php.

1. Ensure the pdb file identifies the higher level structures: Chains (level 2) separated by TER, and Complexes (level 3) separated by REMARK END-OF-COMPLEX:
   ```
   ...
   ATOM ...
   TER (end of chain)
   ATOM ...
   ...
   ATOM ...
   TER (end of chain)
   REMARK END-OF-COMPLEX
   ATOM ...
   ```
2. Execute hcp_getpdb to generate prmtop entries for HCP: hcp_getpdb pdb-filename hcp-prmtop
3. Concatenate the HCP prmtop entries to the end of the standard prmtop file generated by Leap: cat prmtop-file hcp-prmtop > new-prmtop
4. Use this new prmtop file in the NAB molecular dynamics scripts instead of the prmtop file generated by Leap

40. NAB: Sample programs

This chapter provides a variety of examples that use the basic NAB functionality described in earlier chapters to solve interesting molecular manipulation problems. Our hope is that the ideas and approaches illustrated here will facilitate construction of similar programs to solve other problems.

40.1. Duplex Creation Functions

nab provides a variety of functions for creating Watson/Crick duplexes. A short description of four of them is given in this section. All four of these functions are written in nab and the details of their implementation is covered in the section **Creating Watson/Crick Duplexes** of the **User Manual**. You should also look at the function fd_helix() to see how to create duplex helices that correspond to fibre-diffraction models. As with the PERL language, "there is more than one way to do it."

molecule bdna(string seq);
string wc_complement(string seq, string rlib, string rlt);
molecule wc_helix(string seq, string rlib, string natype, string cseq, string crlib,
  string cnatype, float xoffset, float incl, float twist, float rise, string options);
molecule dg_helix(string seq, string rlib, string natype,
  string cseq, string crlib, string cnatype, float xoffset, float incl, float twist, float rise,
  string options);
molecule wc_basepair(residue res, residue cres);

bdna() converts the character string seq containing one or more A, C, G or Ts (or their lower case equivalents) into a uniform ideal Watson/Crick B-form DNA duplex. Each basepair has an X-offset of 2.25 Å, an inclination of -4.96 Å and a helical step of 3.38 Å rise and 36.0o twist. The first character of seq is the 5' base of the strand "sense" of the molecule returned by bdna(). The other strand is called "anti". The phosphates of the two 5' bases have been replaced by hydrogens and and hydrogens have been added to the two O3' atoms of the three prime bases. bdna() returns NULL if it can not create the molecule.

wc_complement() returns a string that is the Watson/Crick complement of its argument seq. Each C, G, T (U) in seq is replaced by G, C and A. The replacements for A depends if rlt is DNA or RNA. If it is DNA, A is replaced by T. If it is RNA A is replaced by U. wc_complement() considers lower case and upper case letters to be the same and always returns upper case letters. wc_complement() returns NULL on error. Note that the while the orientations of the argument string and the returned string are opposite, their absolute orientations are *undefined* until they are used to create a molecule.

wc_helix() creates a uniform duplex from its arguments. The two strands of the returned molecule are called "sense" and "anti". The two sequences, seq and cseq must specify Watson/Crick base pairs. Note the that must be specified as *lower-case* strings, such as "ggact". The nucleic acid type (DNA or RNA) of the sense strand is specified by natype and of the complementary strand cseq by cnatype. Two residue libraries—rlib and crlib—permit creation of DNA:RNA heteroduplexes. If either seq or cseq (but not both) is NULL only the specified strand of what would have been a uniform duplex is created. The options string contains some combination of the strings "s5", "s3", "a5" and "a3"; these indicate which (if any) of the ends of the helices should be "capped" with hydrogens attached to the O5' atom (in place of a phosphate) if "s5" or "a5" is specified, and a proton added to the O3' position if "s3" or "a3" is specified. A blank string indicates no capping, which would be appropriate if this section of helix were to be inserted into a larger molecule. The string "s5a5s3a3" would cap the 5' and 3' ends of both the "sense" and "anti" strands, leading to a chemically complete molecule. wc_helix() returns NULL on error.

dg_helix() is the functional equivalent of wc_helix() but with the backbone geometry minimized via a distance constraint error function. dg_helix() takes the same arguments as wc_helix().

wc_basepair() assembles two nucleic acid residues (assumed to be in a standard orientation) into a two stranded molecule containing one Watson/Crick base pair. The two strands of the new molecule are "sense" and "anti". It returns NULL on error.

40.2. nab and Distance Geometry

Distance geometry is a method which converts a molecule represented as a set of interatomic distances and related information into a 3-D structure. nab has several builtin functions that are used together to provide metric matrix distance geometry. nab also provides the bounds type for holding a molecule's distance geometry information. A bounds object contains the molecule's interatomic distance bounds matrix and a list of its chiral centers and their volumes. nab uses chiral centers with a volume of 0 to enforce planarity.

Distance geometry has several advantages. It is unique in its power to create structures from very incomplete descriptions. It easily incorporates "low resolution structural data" such as that derived from chemical probing since these kinds of experiments generally return only distance bounds. And it also provides an elegant method by which structures may be described functionally.

The nab distance geometry package is described more fully in the section **NAB Language Reference**. Generally, the function newbounds() creates and returns a bounds object corresponding to the molecule mol. This object contains two things—a distance bounds matrix containing initial upper and lower bounds for every pair of atoms in mol and a initial list of the molecules chiral centers and their volumes. Once a bounds object has been initialized, the modeller uses functions from the middle of the distance geometry function list to tighten, loosen or set other distance bounds and chiralities that correspond to experimental measurements or parts of the model's hypothesis. The four functions andbounds(), orbounds(), setbounds and useboundsfrom() work in similar fashion. Each uses two atom expressions to select pairs of atoms from mol. In andbounds(), the current distance bounds of each pair are compared against lb and ub and are replaced by lb, ub if they represent tighter bounds. orbounds() replaces the current bounds of each selected pair, if lb, ub represent looser bounds. setbounds() sets the bounds of all selected pairs to lb, ub. useboundsfrom() sets the bounds between each atom selected in the first expression to a percentage of the distance between the atoms selected in the second atom expression. If the two atom expressions select the same atoms from the same molecule, the bounds between all the atoms selected will be constrained to the current geometry. setchivol() takes four atom expressions that must select exactly four atoms and sets the volume of the tetrahedron enclosed by those atoms to vol. Setting vol to 0 forces those atoms to be planar. getchivol() returns the chiral volume of the tetrahedron described by the four points.

After all experimental and model constraints have been entered into the bounds object, the function tsmooth() applies a process called "triangle smoothing" to them. This tests each triple of distance bounds to see if they can form a triangle. If they can not form a triangle then the distance bounds do not even represent a Euclidean object let alone a 3-D one. If this occurs, tsmooth() quits and returns a 1 indicating failure. If all triples can form triangles, tsmooth() returns a 0. Triangle smoothing pulls in the large upper bounds. After all, the maximum distance between two atoms can not exceed the sum of the upper bounds of the shortest path between them. Triangle smoothing can also increase lower bounds, but this process is much less effective as it requires one or more large lower bounds to begin with.

The function embed() takes the smoothed bounds and converts them into a 3-D object. This process is called "embedding". It does this by choosing a random distance for each pair of atoms within the bounds of that pair. Sometimes the bounds simply do not represent a 3-D object and embed() fails, returning the value 1. This is rare and usually indicates the that the distance bounds matrix part of the bounds object contains errors. If the distance set does embed, conjgrad() can subject newly embedded coordinates to conjugate gradient refinement against the distance and chirality information contained in bounds. The refined coordinates can replace the current coordinates of the molecule in mol. embed() returns a 0 on success and conjgrad() returns an exit code explained further in the **Language Reference** section of this manual. The call to embed() is usually placed in a loop with each new structure saved after each call to see the diversity of the structures the bounds represent.

In addition to the explicit bounds manipulation functions, nab provides an implicit way of setting bounds between interacting residues. The function setboundsfromdb() is for use in creating distance and chirality bounds for nucleic acids. setboundsfromdb() takes as an argument two atom expressions selecting two residues, the name of a database containing bounds information, and a number which dictates the tightness of the bounds. For instance, if the database *bdna.stack.db* is specified, setboundsfromdb() sets the bounds between the two residues to what they would be if they were stacked in strand in a typical Watson-Crick B-form duplex. Similarly, if the database *arna.basepair.db* is specified, setboundsfromdb() sets the bounds between the two residues to what they would be if the two residues form a typical Watson-Crick basepair in an A-form helix.

40.2.1. Refine DNA Backbone Geometry

As mentioned previously, wc_helix() performs rigid body transformations on residues and does not correct for poor backbone geometry. Using distance geometry, several techniques are available to correct the backbone geometry. In program 7, an 8-basepair dna sequence is created using wc_helix(). A new bounds object is created on line 14, which automatically sets all the 1-2, 1-3, and 1-4 distance bounds information according the geometry of the model. Since this molecule was created using wc_helix(), the O3'-P distance between adjacent stacked residues is often not the optimal 1.595 °, and hence, the 1-2, 1-3, and 1-4, distance bounds set by newbounds() are incorrect. We want to preserve the position of the nucleotide bases, however, since this is the helix whose backbone we wish to minimize. Hence the call to useboundsfrom() on line 17 which sets the bounds from every atom in each nucleotide base to the actual distance to every other atom in every other nucleotide base. *In general, the likelihood of a distance geometry refinement to satisfy a given bounds criteria is proportional to the number of (consistent) bounds set supporting that criteria*. In other words, the more bounds that are set supporting a given conformation, the greater the chance that conformation will resolve after the refinement. An example of this concept is the use of useboundsfrom() in line 17, which works to preserve our rigid helix conformation of all the nucleotide base atoms.

We can correct the backbone geometry by overwriting the erroneous bounds with more appropriate bounds. In lines 19-29, all the 1-2, 1-3, and 1-4 bounds involving the O3'-P connection between strand 1 residues are set to that which would be appropriate for an idealized phosphate linkage. Similarly, in lines 31-41, all the 1-2, 1-3, and 1-4 bounds involving the O3'-P connection among strand 2 residues are set to an idealized conformation. This technique is effective since all the 1-2, 1-3, and 1-4 distance bounds created by newbounds() include those of the idealized nucleotides in the nucleic acid libraries dna.amber94.rlb, rna.amber94.rlb, *etc.* contained in reslib. Hence, by setting these bounds and refining against the distance energy function, we are spreading the 'error' across the backbone, where the 'error' is the departure from the idealized sugar conformation and idealized phosphate linkage.

On line 43, we smooth the bounds matrix, and on line 44 we give a substantial penalty for deviating from a 3-D refinement by setting k4d=4.0. Notice that there is no need to embed the molecule in this program, as the actual coordinates are sufficient for any refinement.

```
1   //  Program 7 - refine backbone geometry using distance function
2   molecule m;
3   bounds b;
4   string  seq, cseq;
5   int i;
6   float xyz[ dynamic ], fret;
7
8   seq = "acgtacgt";
9   cseq = wc_complement( "acgtacgt", "", "dna" );
10
11  m = wc_helix( seq, "", "dna", cseq, "",
12                "dna", 2.25, -4.96, 36.0, 3.38, "" );
13
14  b = newbounds(m, "");
15  allocate xyz[ 4*m.natoms ];
16
17  useboundsfrom(b, m, "::??,H?[^T']", m, "::??,H?[^T']", 0.0 );
18  for ( i = 1; i < m.nresidues/2 ; i = i + 1 ){
19          setbounds(b,m, sprintf("1:%d:O3'",i),
20                         sprintf("1:%d:P",i+1),   1.595,1.595);
21          setbounds(b,m, sprintf("1:%d:O3'",i),
22                         sprintf("1:%d:O5'",i+1), 2.469,2.469);
```

```
        setbounds(b,m, sprintf("1:%d:C3'",i),
                       sprintf("1:%d:P",i+1),   2.609,2.609);
        setbounds(b,m, sprintf("1:%d:O3'",i),
                       sprintf("1:%d:O1P",i+1), 2.513,2.513);
        setbounds(b,m, sprintf("1:%d:O3'",i),
                       sprintf("1:%d:O2P",i+1), 2.515,2.515);
        setbounds(b,m, sprintf("1:%d:C4'",i),
                       sprintf("1:%d:P",i+1),   3.550,4.107);
        setbounds(b,m, sprintf("1:%d:C2'",i),
                       sprintf("1:%d:P",i+1),   3.550,4.071);
        setbounds(b,m, sprintf("1:%d:C3'",i),
                       sprintf("1:%d:O1P",i+1), 3.050,3.935);
        setbounds(b,m, sprintf("1:%d:C3'",i),
                       sprintf("1:%d:O2P",i+1), 3.050,4.004);
        setbounds(b,m, sprintf("1:%d:C3'",i),
                       sprintf("1:%d:O5'",i+1), 3.050,3.859);
        setbounds(b,m, sprintf("1:%d:O3'",i),
                       sprintf("1:%d:C5'",i+1), 3.050,3.943);

        setbounds(b,m, sprintf("2:%d:P",i+1),
                       sprintf("2:%d:O3'",i),   1.595,1.595);
        setbounds(b,m, sprintf("2:%d:O5'",i+1),
                       sprintf("2:%d:O3'",i),   2.469,2.469);
        setbounds(b,m, sprintf("2:%d:P",i+1),
                       sprintf("2:%d:C3'",i),   2.609,2.609);
        setbounds(b,m, sprintf("2:%d:O1P",i+1),
                       sprintf("2:%d:O3'",i),   2.513,2.513);
        setbounds(b,m, sprintf("2:%d:O2P",i+1),
                       sprintf("2:%d:O3'",i),   2.515,2.515);
        setbounds(b,m, sprintf("2:%d:P",i+1),
                       sprintf("2:%d:C4'",i),   3.550,4.107);
        setbounds(b,m, sprintf("2:%d:P",i+1),
                       sprintf("2:%d:C2'",i),   3.550,4.071);
        setbounds(b,m, sprintf("2:%d:O1P",i+1),
                       sprintf("2:%d:C3'",i),   3.050,3.935);
        setbounds(b,m, sprintf("2:%d:O2P",i+1),
                       sprintf("2:%d:C3'",i),   3.050,4.004);
        setbounds(b,m, sprintf("2:%d:O5'",i+1),
                       sprintf("2:%d:C3'",i),   3.050,3.859);
        setbounds(b,m, sprintf("2:%d:C5'",i+1),
                       sprintf("2:%d:O3'",i),   3.050,3.943);
}
tsmooth( b, 0.0005 );
dg_options(b, "seed=33333, gdist=0, ntpr=100, k4d=4.0" );
setxyzw_from_mol( m, NULL, xyz );
conjgrad( xyz, 4*m.natoms, fret, db_viol, 0.1, 10., 500 );
setmol_from_xyzw( m, NULL, xyz );
putpdb( "acgtacgt.pdb", m );
```

The approach of Program 7 is effective but has a disadvantage in that it does not scale linearly with the number of atoms in the molecule. In particular, tsmooth() and conjgrad() require extensive CPU cycles for large numbers of residues. For this reason, the function dg_helix() was created. dg_helix() takes uses the same method of Program 7, but employs a 3-basepair helix template which traverses the new helix as it is being constructed. In this way, the helix is built in a piecewise manner and the maximum number of residues considered in each refinement is less than or equal to six. This is the preferred method of helix construction for large, idealized canonical duplexes.

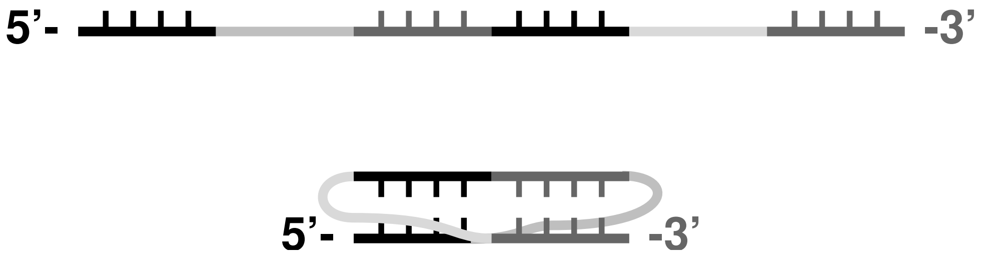


Figure 40.1.: *Single-stranded RNA* (top) *folded into a pseudoknot* (bottom). *The black and dark grey base pairs can be stacked.*

40.2.2. RNA Pseudoknots

In addition to the standard helix generating functions, nab provides extensive support for generating initial structures from low structural information. As an example, we will describe the construction of a model of an RNA pseudoknot based on a small number of secondary and tertiary structure descriptions. Shen and Tinoco (*J. Mol. Biol.* **247,** 963-978, 1995) used the molecular mechanics program X-PLOR to determine the three dimensional structure of a 34 nucleotide RNA sequence that folds into a pseudoknot. This pseudoknot promotes frame shifting in Mouse Mammary Tumor Virus. A pseudoknot is a single stranded nucleic acid molecule that contains two improperly nested hairpin loops as shown in Figure 40.1. NMR distance and angle constraints were converted into a three dimensional structure using a two stage restrained molecular dynamics protocol. Here we show how a three-dimensional model can be constructed using just a few key features derived from the NMR investigation.

```
// Program 8 - create a pseudoknot using distance geometry
molecule m;
float   xyz[ dynamic ],f[ dynamic ],v[ dynamic ];
bounds  b;
int     i, seqlen;
float   fret;
string  seq, opt;

seq = "GCGGAAACGCCGCGUAAGCG";

seqlen = length(seq);

m = link_na("1", seq, "", "RNA", "35");

allocate xyz[ 4*m.natoms ];
allocate f[ 4*m.natoms ];
allocate v[ 4*m.natoms ];

b = newbounds(m, "");

for ( i = 1; i <= seqlen; i = i + 1) {
        useboundsfrom(b, m, sprintf("1:%d:??,H?[^'T]", i), m,
        sprintf("1:%d:??,H?[^'T]", i), 0.0 );
}

setboundsfromdb(b, m, "1:1:",  "1:2:",  "arna.stack.db",    1.0);
setboundsfromdb(b, m, "1:2:",  "1:3:",  "arna.stack.db",    1.0);
setboundsfromdb(b, m, "1:3:",  "1:18:", "arna.stack.db",    1.0);
setboundsfromdb(b, m, "1:18:", "1:19:", "arna.stack.db",    1.0);
setboundsfromdb(b, m, "1:19:", "1:20:", "arna.stack.db",    1.0);

setboundsfromdb(b, m, "1:8:",  "1:9:",  "arna.stack.db",    1.0);
```

```
33  setboundsfromdb(b, m, "1:9:",  "1:10:", "arna.stack.db",    1.0);
34  setboundsfromdb(b, m, "1:10:", "1:11:", "arna.stack.db",    1.0);
35  setboundsfromdb(b, m, "1:11:", "1:12:", "arna.stack.db",    1.0);
36  setboundsfromdb(b, m, "1:12:", "1:13:", "arna.stack.db",    1.0);
37
38  setboundsfromdb(b, m, "1:1:",  "1:13:", "arna.basepair.db", 1.0);
39  setboundsfromdb(b, m, "1:2:",  "1:12:", "arna.basepair.db", 1.0);
40  setboundsfromdb(b, m, "1:3:",  "1:11:", "arna.basepair.db", 1.0);
41
42  setboundsfromdb(b, m, "1:8:",  "1:20:", "arna.basepair.db", 1.0);
43  setboundsfromdb(b, m, "1:9:",  "1:19:", "arna.basepair.db", 1.0);
44  setboundsfromdb(b, m, "1:10:", "1:18:", "arna.basepair.db", 1.0);
45
46  tsmooth(b, 0.0005);
47
48  opt = "seed=571, gdist=0, ntpr=50, k4d=2.0, randpair=5., sqviol=1";
49  dg_options( b, opt );
50  embed(b, xyz );
51
52  conjgrad( xyz, 4*m.natoms, fret, db_viol, 0.1, 10., 500 );
53
54  setmol_from_xyzw( m, NULL, xyz );
55  putpdb( "rna_pseudoknot.pdb", m );
```

Program 8 uses distance geometry followed by minimization and simulated annealing to create a model of a pseudoknot. Distance geometry code begins in line 20 with the call to newbounds() and ends on line 53 with the call to embed(). The structure created with distance geometry is further refined with molecular dynamics in lines 58-74. Note that very little structural information is given - only connectivity and general base-base interactions. The stacking and base-pair interactions here are derived from NMR evidence, but in other cases might arise from other sorts of experiments, or as a model hypothesis to be tested.

The 20-base RNA sequence is defined on line 9. The molecule itself is created with the link_na() function call which creates an extended conformation of the RNA sequence and caps the 5' and 3' ends. Lines 15-17 define arrays that will be used in the simulated annealing of the structure. The bounds object is created in line 19 which automatically sets the 1-2, 1-3, and 1-4 distance bounds in the molecule. The loop in lines 21-24 sets the bounds of each atom in each residue base to the actual distance to every other atom in the same base. This has the effect of enforcing the planarity of the base by treating the base somewhat like a rigid body. In lines 26-44, bounds are set according to information stored in a database. The setboundsfromdb() call sets the bounds from all the atoms in the two specified residues to a 1.0 multiple of the standard deviation of the bounds distances in the specified database. Specifically, line 26 sets the bounds between the base atoms of the first and second residues of strand 1 to be within one standard deviation of a *typical* aRNA stacked pair. Similarly, line 38 sets the bounds between residues 1 and 13 to be that of *typical* Watson-Crick basepairs. For a description of the setboundsfromdb() function, see Chapter 1.

Line 46 smooths the bounds matrix, by attempting to adjust any sets of bounds that violate the triangle equality. Lines 48-49 initialize some distance geometry variables by setting the random number generator seed, declaring the type of distance distribution, how often to print the energy refinement process, declaring the penalty for using a 4th dimension in refinement, and which atoms to use to form the initial metric matrix. The coordinates are calculated and embedded into a 3D coordinate array, *xyz* by the embed() function call on line 50.

The coordinates *xyz* are subject to conjugate gradient refinements in line 52. Line 54 replaces the old molecular coordinates with the new refined ones, and lastly, on line 55, the molecule is saved as "rna_pseudoknot.pdb".

The resulting structure of Program 8 is shown in Figure 40.2. This structure had an final total energy of 46 units. The helical region, shown as polytubes, shows stacking and wc-pairing interactions and a well-defined right-handed helical twist. Of course, good modeling of a "real" pseudoknot would require putting in more constraints, but this example should illustrate how to get started on problems like this.

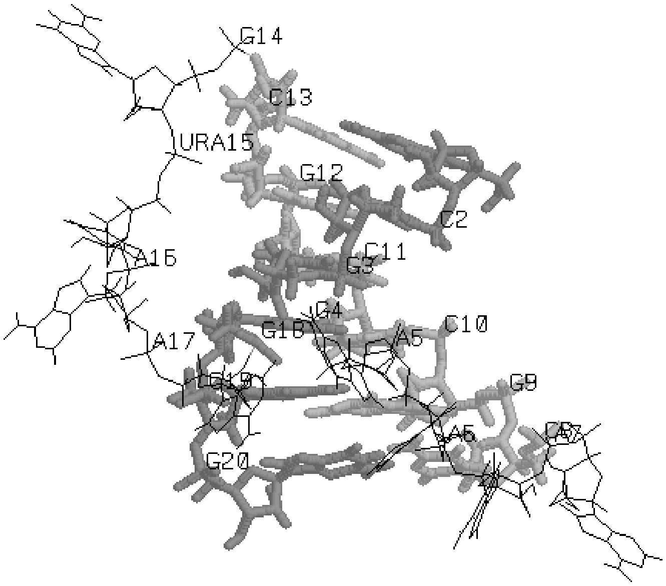


Figure 40.2.: *Folded RNA pseudoknot.*

40.2.3. NMR refinement for a protein

Distance geometry techniques are often used to create starting structures in NMR refinement. Here, in addition to the covalent connections, one makes use of a set of distance and torsional restraints derived from NMR data. While NAB is not (yet?) a fully-functional NMR refinement package, it has enough capabilities to illustrate the basic ideas, and could be the starting point for a flexible procedure. Here we give an illustration of how the rough structure of a protein can be determined using distance geometry and NMR distance constraints; the structures obtained here would then be candidates for further refinement in programs like X-plor or Amber.

The program below illustrates a general procedure for a primarily helical DNA binding domain. Lines 15-22 just construct the sequence in an extended conformation, such that bond lengths and angles are correct, but none of the torsions are correct. The bond lengths and angles are used by newbounds() to construct the "covalent" part of the bounds matrix.

```
1  //  Program 8a. General driver routine to do distance geometry \fC
2  //    on proteins, with DYANA-like distance restraints.\fC
3
4  #define MAXCOORDS 12000
5
6  molecule m;
7  atom    a;
8  bounds    b;
9  int    ier,i, numstrand, ires,jres;
10 float    fret, rms, ub;
11 float    xyz[ MAXCOORDS ], f[ MAXCOORDS ], v[ MAXCOORDS ];
12 file    boundsf;
13 string  iresname,jresname,iat,jat,aex1,aex2,aex3,aex4,line,dgopts,seq;
14
15 //  sequence of the mrf2 protein:
16 seq = "RADEQAFLVALYKYMKERKTPIERIPYLGFKQINLWTMFQAAQKLGGYETITARRQWKHIY"
17     + "DELGGNPGSTSAATCTRRHYERLILPYERFIKGEEDKPLPPIKPRK";
```

```

//  build this sequence in an extended conformation, and construct a bounds
//     matrix just based on the covalent structure:
m = linkprot( "A", seq, "" );
b = newbounds( m, "" );

//  read in constraints, updating the bounds matrix using "andbounds":

//  distance constraints are basically those from Y.-C. Chen, R.H.  Whitson
//    Q. Liu, K. Itakura and Y. Chen, "A novel DNA-binding motif shares
//    structural homology to DNA replication and repair nucleases and
//    polymerases,"  Nature Sturct. Biol. 5:959-964 (1998).

boundsf = fopen( "mrf2.7col", "r" );
while( line = getline( boundsf ) ){
    sscanf( line, "%d %s %s %d %s %s %lf", ires, iresname, iat,
        jres, jresname, jat, ub );

//   translations for DYANA-style pseudoatoms:
    if( iat == "HN" ){ iat = "H"; }
    if( jat == "HN" ){ jat = "H"; }

    if( iat == "QA" ){ iat = "CA"; ub += 1.0; }
    if( jat == "QA" ){ jat = "CA"; ub += 1.0; }
    if( iat == "QB" ){ iat = "CB"; ub += 1.0; }
    if( jat == "QB" ){ jat = "CB"; ub += 1.0; }
    if( iat == "QG" ){ iat = "CG"; ub += 1.0; }
    if( jat == "QG" ){ jat = "CG"; ub += 1.0; }
    if( iat == "QD" ){ iat = "CD"; ub += 1.0; }
    if( jat == "QD" ){ jat = "CD"; ub += 1.0; }
    if( iat == "QE" ){ iat = "CE"; ub += 1.0; }
    if( jat == "QE" ){ jat = "CE"; ub += 1.0; }
    if( iat == "QQG" ){ iat = "CB"; ub += 1.8; }
    if( jat == "QQG" ){ jat = "CB"; ub += 1.8; }
    if( iat == "QQD" ){ iat = "CG"; ub += 1.8; }
    if( jat == "QQD" ){ jat = "CG"; ub += 1.8; }
    if( iat == "QG1" ){ iat = "CG1"; ub += 1.0; }
    if( jat == "QG1" ){ jat = "CG1"; ub += 1.0; }
    if( iat == "QG2" ){ iat = "CG2"; ub += 1.0; }
    if( jat == "QG2" ){ jat = "CG2"; ub += 1.0; }
    if( iat == "QD1" ){ iat = "CD1"; ub += 1.0; }
    if( jat == "QD1" ){ jat = "CD1"; ub += 1.0; }
    if( iat == "QD2" ){ iat = "ND2"; ub += 1.0; }
    if( jat == "QD2" ){ jat = "ND2"; ub += 1.0; }
    if( iat == "QE2" ){ iat = "NE2"; ub += 1.0; }
    if( jat == "QE2" ){ jat = "NE2"; ub += 1.0; }

    aex1 = ":" + sprintf( "%d", ires) + ":" + iat;
    aex2 = ":" + sprintf( "%d", jres) + ":" + jat;
    andbounds( b, m, aex1, aex2, 0.0, ub );
}
fclose( boundsf );

//  add in helical chirality constraints to force right-handed helices:
//    (hardwire in locations 1-16, 36-43, 88-92)
for( i=1; i<=12; i++){
    aex1 = ":" + sprintf( "%d", i ) + ":CA";
    aex2 = ":" + sprintf( "%d", i+1 ) + ":CA";
    aex3 = ":" + sprintf( "%d", i+2 ) + ":CA";
    aex4 = ":" + sprintf( "%d", i+3 ) + ":CA";
    setchivol( b, m, aex1, aex2, aex3, aex4, 7.0 );
}
for( i=36; i<=39; i++){
    aex1 = ":" + sprintf( "%d", i ) + ":CA";
    aex2 = ":" + sprintf( "%d", i+1 ) + ":CA";
    aex3 = ":" + sprintf( "%d", i+2 ) + ":CA";
    aex4 = ":" + sprintf( "%d", i+3 ) + ":CA";
    setchivol( b, m, aex1, aex2, aex3, aex4, 7.0 );
}
for( i=88; i<=89; i++){
    aex1 = ":" + sprintf( "%d", i ) + ":CA";
    aex2 = ":" + sprintf( "%d", i+1 ) + ":CA";
    aex3 = ":" + sprintf( "%d", i+2 ) + ":CA";
    aex4 = ":" + sprintf( "%d", i+3 ) + ":CA";
    setchivol( b, m, aex1, aex2, aex3, aex4, 7.0 );
}

//  set up some options for the distance geometry calculation
//   here use the random embed method:
dgopts = "ntpr=10000,rembed=1,rbox=300.,riter=250000,seed=8511135";
dg_options( b, dgopts );

//  do triangle-smoothing on the bounds matrix, then embed:
geodesics( b ); embed( b, xyz );

//  now do conjugate-gradient minimization on the resulting structures:

//  first, weight the chirality constraints heavily:
dg_options( b, "ntpr=20, k4d=5.0, sqviol=0, kchi=50." );
conjgrad( xyz, 4*m.natoms, fret, db_viol, 0.02, 1000., 300 );

//  next, squeeze out the fourth dimension, and increase penalties for
//     distance violations:
dg_options( b, "k4d=10.0, sqviol=1, kchi=50." );
conjgrad( xyz, 4*m.natoms, fret, db_viol, 0.02, 100., 400 );

//  transfer the coordinates from the "xyz" array to the molecule
//  itself, and print out the violations:
setmol_from_xyzw( m, NULL, xyz );
dumpboundsviolations( stdout, b, 0.5 );

//  do a final short molecular-mechanics "clean-up":
putpdb( m, "temp.pdb" );
m = getpdb_prm( "temp.pdb", "leaprc.protein.ff14SB", "", 0 );
setxyz_from_mol( m, NULL, xyz );

mm_options( "cut=10.0" );
mme_init( m, NULL, "::ZZZ", xyz, NULL );
conjgrad( xyz, 3*m.natoms, fret, mme, 0.02, 100., 200 );
setmol_from_xyz( m, NULL, xyz );
putpdb( argv[3] + ".mm.pdb", m );
```

Once the covalent bounds are created, the the bounds matrix is modified by constraints constructed from an NMR analysis program. This particular example uses the format of the DYANA program, but NAB could be easily modified to read in other formats as well. Here are a few lines from the *mrf2.7col* file:

```
1 ARG+ QB   2 ALA  QB  7.0
4 GLU- HA 93 LYS+ QB  7.0
5 GLN  QB   8 LEU  QQD 9.9
5 GLN  HA   9 VAL  QQG 6.4
85 ILE HA 92 ILE  QD1 6.0
5 GLN  HN   1 ARG+ O   2.0
5 GLN  N    1 ARG+ O   3.0
6 ALA  HN   2 ALA  O   2.0
6 ALA  N    2 ALA  O   3.0
```

The format should be self-explanatory, with the final number giving the upper bound. Code in lines 31-69 reads these in, and translates pseudo-atom codes like "QQD" into atom names. Lines 71-93 add in chirality constraints to ensure right-handed alpha-helices: distance constraints alone do not distinguish chirality, so additions like this are often necessary. The "actual" distance geometry steps take place in line 101, first by triangle-smoothing the bounds, then by embedding them into a three-dimensional object. The structures at this point are actually generally quite bad, so "real-space" refinement is carried out in lines 103-112, and a final short molecular mechanics minimization in lines 119-126.

It is important to realize that many of the structures for the above scheme will get "stuck", and not lead to good structures for the complex. Helical proteins are especially difficult for this sort of distance geometry, since helices (or even parts of helices) start out left-handed, and it is not always possible to easily convert these to right-handed structures. For this particular example, (using different values for the *seed* in line 97), we find that about 30-40% of the structures are "acceptable", in the sense that further refinement in Amber yields good structures.

40.3. Building Larger Structures

While the DNA duplex is locally rather stiff, many DNA molecules are sufficiently long that they can be bent into a wide variety of both open and closed curves. Some examples would be simple closed circles, supercoiled closed circles that have relaxed into circles with twists and the nucleosome core fragment where the duplex itself is wound into a short helix. This section shows how `nab` can be used to “wrap” DNA around a curve. Three examples are provided: the first produces closed circles with or without supercoiling, the second creates a simple model of the nucleosome core fragment and the third shows how to wind a duplex around a more arbitrary open curve specified as a set of points. The examples are fairly general but do require that the curves be relatively smooth so that the deformation from a linear duplex at each step is small.

Before discussing the examples and the general approach they use, it will be helpful to define some terminology. The helical axis of a base pair is the helical axis defined by an ideal B-DNA duplex that contains that base pair. The base pair plane is the mean plane of both bases. The origin of a base pair is at the intersection the base pair’s helical axis and its mean plane. Finally the rise is the distance between the origins of adjacent base pairs.

The overall strategy for wrapping DNA around a curve is to create the curve, find the points on the curve that contain the base pair origins, place the base pairs at these points, oriented so that their helical axes are tangent to the curve and finally rotate the base pairs so that they have the correct helical twist. In all the examples below, the points are chosen so that the rise is constant. This is by no means an absolute requirement, but it does simplify the calculations needed to locate base pairs, and is generally true for the gently bending curves these examples are designed for. In examples 1 and 2, the curve is simple, either a circle or a helix, so the points that locate the base pairs are computed directly. In addition, the bases are rotated about their original helical axes so that they have the correct helical orientation before being placed on the curve.

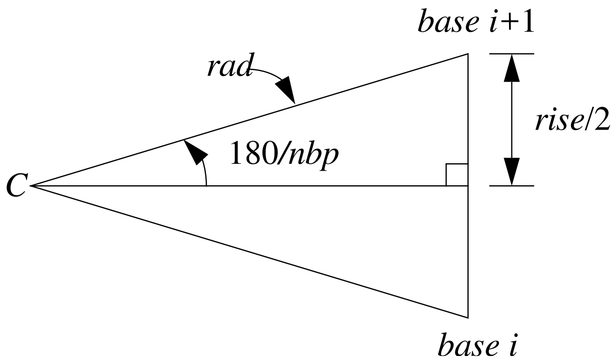
However, this method is inadequate for the more complicated curves that can be handled by example 3. Here each base is placed on the curve so that its helical axis is aligned correctly, but its helical orientation with respect to the previous base is arbitrary. It is then rotated about its helical axis so that it has the correct twist with respect to the previous base.

40.3.1. Closed Circular DNA

This section describes how to use nab to make closed circular duplex DNA with a uniform rise of 3.38 Å. Since the distance between adjacent base pairs is fixed, the radius of the circle that forms the axis of the duplex depends only on the number of base pairs and is given by this rule:

$$rad = rise/(2\sin(180/nbp))$$

where *nbp* is the number of base pairs. To see why this is so, consider the triangle below formed by the center of the circle and the centers of two adjacent base pairs. The two long sides are radii of the circle and the third side is the rise. Since the the base pairs are uniformly distributed about the circle the angle between the two radii is 360/*nbp*. Now consider the right triangle in the top half of the original triangle. The angle at the center is 180/*nbp*, the opposite side is *rise/2* and *rad* follows from the definition of sin.



In addition to the radius, the helical twist which is a function of the amount of supercoiling must also be computed. In a closed circular DNA molecule, the last base of the duplex must be oriented in such a way that a single helical step will superimpose it on the first base. In circles based on ideal B-DNA, with 10 bases/turn, this requires that the number of base pairs in the duplex be a multiple of 10. Supercoiling adds or subtracts one or more whole turns. The amount of supercoiling is specified by the $\Delta linkingnumber$ which is the number of extra turns to add or subtract. If the original circle had *nbp*/10 turns, the supercoiled circle will have $nbp/10+\Delta lk$ turns. As each turn represents 360o of twist and there are *nbp* base pairs, the twist between base pairs is

$$(nbp/10+\Delta lk)\times 360/nbp$$

At this point, we are ready to create models of circular DNA. Bases are added to model in three stages. Each base pair is created using the nab builtin wc_helix(). It is originally in the XY plane with its center at the origin. This makes it convenient to create the DNA circle in the XZ plane. After the base pair has been created, it is rotated around its own helical axis to give it the proper twist, translated along the global X axis to the point where its center intersects the circle and finally rotated about the Y axis to move it to its final location. Since the first base pair would be both twisted about Z and rotated about Y 0o, those steps are skipped for base one. A detailed description follows the code.

```
// Program 9 - Create closed circular DNA.
#define RISE    3.38

int     b, nbp, dlk;
float       rad, twist, ttw;
molecule    m, m1;
matrix      matdx, mattw, matry;
string      sbase, abase;
int     getbase();

if( argc != 3 ){
    fprintf( stderr, "usage: %s nbp dlk\\n", argv[ 1 ] );
```

```
        exit( 1 );
}

nbp = atoi( argv[ 2 ] );
if( !nbp || nbp % 10 ){
    fprintf( stderr,
    "%s: Num. of base pairs must be multiple of 10\\n",
        argv[ 1 ] );
    exit( 1 );
}

dlk = atoi( argv[ 3 ] );

twist = ( nbp / 10 + dlk ) * 360.0 / nbp;
rad = 0.5 * RISE / sin( 180.0 / nbp );

matdx = newtransform( rad, 0.0, 0.0, 0.0, 0.0, 0.0 );

m = newmolecule();
addstrand( m, "A" );
addstrand( m, "B" );
ttw = 0.0;
for( b = 1; b <= nbp; b = b + 1 ){

    getbase( b, sbase, abase );

    m1 = wc_helix(
        sbase, "", "dna", abase, "",
        "dna", 2.25, -4.96, 0.0, 0.0 );

    if( b > 1 ){
        mattw = newtransform( 0.,0.,0.,0.,0.,ttw );
        transformmol( mattw, m1, NULL );
    }

    transformmol( matdx, m1, NULL );

    if( b > 1 ){
        matry = newtransform(
            0.,0.,0.,0.,-360.*(b-1)/nbp,0. );
        transformmol( matry, m1, NULL );
    }

    mergestr( m, "A", "last", m1, "sense", "first" );
    mergestr( m, "B", "first", m1, "anti", "last" );
    if( b > 1 ){
        connectres( m, "A", b - 1, "O3'", b, "P" );
        connectres( m, "B", 1, "O3'", 2, "P" );
    }

    ttw = ttw + twist;
    if( ttw >= 360.0 )
        ttw = ttw - 360.0;
}

connectres( m, "A", nbp, "O3'", 1, "P" );
connectres( m, "B", nbp, "O3'", 1, "P" );
```

```
70
71 putpdb( "circ.pdb", m );
72 putbnd( "circ.bnd", m );
```

The code requires two integer arguments which specify the number of base pairs and the$\Delta linkingnumber$or the amount of supercoiling. Lines 11-24 process the arguments making sure that they conform to the model's assumptions. In lines 11-14, the code checks that there are exactly three arguments (the nab program's name is argument one), and exits with a error message if the number of arguments is different. Next lines 16-22 set the number of base pairs (nbp) and test to make certain it is a nonzero multiple of 10, again exiting with an error message if it is not. Finally the $\Delta linkingnumber$(dlk) is set in line 24. The helical twist and circle radius are computed in lines 26 and 27 in accordance with the formulas developed above. Line 29 creates a transformation matrix, matdx, that is used to move each base from the global origin along the X-axis to the point where its center intersects the circle.

The circular DNA is built in the molecule variable m, which is initialized and given two strands, "A" and "B" in lines 30-32. The variable ttw in line 34 holds the total twist applied to each base pair The molecule is created in the loop from lines 35-66. The base pair number (b) is converted to the appropriate strings specifying the two nucleotides in this pair. This is done by the function getbase(). This source of this function must be provided by the user who is creating the circles as only he or she will know the actual DNA sequence of the circle. Once the two bases are specified they are passed to the nab builtin wc_helix() which returns a single base pair in the XY plane with its center at the origin. The helical axis of this base pair is on the Z-axis with the 5'-3' direction oriented in the positive Z-direction.

One or three transformations is required to position this base in its correct place in the circle. It must be rotated about the Z-axis (its helical axis) so that it is one additional unit of twist beyond the previous base. This twist is done in lines 43-46. Since the first base needs 0o twist, this step is skipped for it. In line 48, the base pair is moved in the positive direction along the X-axis to place the base pair's origin on the circle. Finally, the base pair is rotated about the Y-axis in lines 50-54 to bring it to its proper position on the circle. Again, since this rotation is 0o for base 1, this step is also skipped for the first base.

In lines 56-57, the newly positioned base pair in m1 is added to the growing molecule in m. Note that since the two strands of DNA are antiparallel, the "sense" strand of m1 is added after the last base of the "A" strand of m and the "anti" strand of m1 is added before the first base of the "B" strand of m. For all but the first base, the newly added residues are bonded to the residues they follow (or precede). This is done by the two calls to connectres() in lines 59-60. Again, due to the antiparallel nature of DNA, the new residue in the "A" strand is residue b, but is residue 1 in the "B" strand. In line 63-65, the total twist (ttw) is updated and adjusted to keep in in the range [0,360). After all base pairs have been added the loop exits.

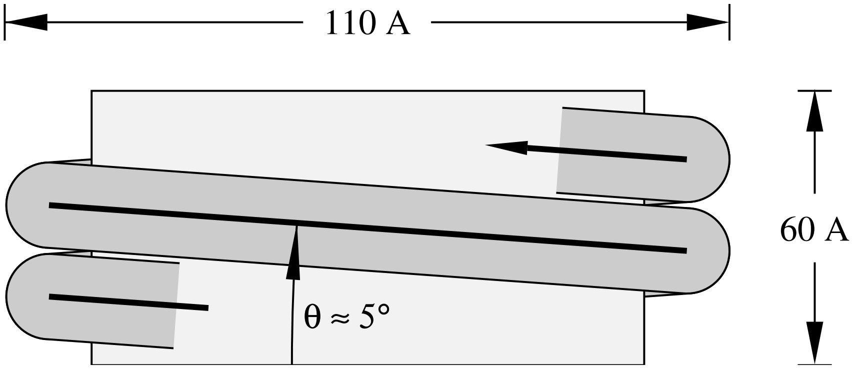
After the loop exit, since this is a *closed* circular molecule the first and last bases of each strand must be bonded and this is done with the two calls to connectres() in lines 67-68. The last step is to save the molecule's coordinates and connectivity in lines 71-72. The nab builtin putpdb() writes the coordinate information in PDB format to the file "circ.pdb" and the nab builtin putbnd() saves the bonding as pairs of integers, one pair/line in the file "circ.bnd", where each integer in a pair refers to an ATOM record in the previously written PDB file.

40.3.2. Nucleosome Model

While the DNA duplex is locally rather stiff, many DNA molecules are sufficiently long that they can be bent into a wide variety of both open and closed curves. Some examples would be simple closed circles, supercoiled closed circles that have relaxed into circles with twists, and the nucleosome core fragment, where the duplex itself is wound into a short helix.

The overall strategy for wrapping DNA around a curve is to create the curve, find the points on the curve that contain the base pair origins, place the base pairs at these points, oriented so that their helical axes are tangent to the curve, and finally rotate the base pairs so that they have the correct helical twist. In the example below, the simplifying assumption is made that the rise is constant at 3.38 Å.

The nucleosome core fragment is composed of duplex DNA wound in a left handed helix around a central protein core. A typical core fragment has about 145 base pairs of duplex DNA forming about 1.75 superhelical turns. Measurements of the overall dimensions of the core fragment indicate that there is very little space between adjacent wraps of the duplex. A side view of a schematic of core particle is shown below.



Computing the points at which to place the base pairs on a helix requires us to spiral an inelastic wire (representing the helical axis of the bent duplex) around a cylinder (representing the protein core). The system is described by four numbers of which only three are independent. They are the number of base pairs n, the number of turns its makes around the protein core t, the "winding" angle θ (which controls how quickly the the helix advances along the axis of the core) and the helix radius r. Both the the number of base pairs and the number of turns around the core can be measured. The leaves two choices for the third parameter. Since the relationship of the winding angle to the overall particle geometry seems more clear than that of the radius, this code lets the user specify the number of turns, the number of base pairs and the winding angle, then computes the helical radius and the displacement along the helix axis for each base pair:

$$d = 3.38\sin(\theta);\ \phi = 360t/(n-1) \tag{40.1}$$

$$r = \frac{3.38(n-1)\cos(\theta)}{2\pi t} \tag{40.2}$$

where d and $\phi$are the displacement along and rotation about the protein core axis for each base pair.

These relationships are easily derived. Let the nucleosome core particle be oriented so that its helical axis is along the global Y-axis and the lower cap of the protein core is in the XZ plane. Consider the circle that is the projection of the helical axis of the DNA duplex onto the XZ plane. As the duplex spirals along the core particle it will go around the circle t times, for a total rotation of $360t$o. The duplex contains $(n-1)$ steps, resulting in $360t/(n-1)$o of rotation between successive base pairs.

```
// Program 10.  Create simple nucleosome model.
#define PI  3.141593
#define RISE    3.38
#define TWIST   36.0
int         b, nbp; int getbase();
float       nt, theta, phi, rad, dy, ttw, len, plen, side;
molecule    m, m1;
matrix      matdx, matrx, maty, matry, mattw;
string      sbase, abase;

nt = atof( argv[ 2 ] );    // number of turns
nbp = atoi( argv[ 3 ] );   // number of base pairs
theta = atof( argv[ 4 ] ); // winding angle

dy = RISE * sin( theta );
phi = 360.0 * nt / ( nbp-1 );
rad = (( nbp-1 )*RISE*cos( theta ))/( 2*PI*nt );

matdx = newtransform( rad, 0.0, 0.0, 0.0, 0.0, 0.0 );
```

```
20 matrx = newtransform( 0.0, 0.0, 0.0, -theta, 0.0, 0.0 );
21 
22 m = newmolecule();
23 addstrand( m, "A" ); addstrand( m, "B" );
24 ttw = 0.0;
25 for( b = 1; b <= nbp; b = b + 1 ){
26     getbase( b, sbase, abase );
27     m1 = wc_helix( sbase, "", "dna", abase, "", "dna",
28         2.25, -4.96, 0.0, 0.0 );
29     mattw = newtransform( 0., 0., 0., 0., 0., ttw );
30     transformmol( mattw, m1, NULL );
31     transformmol( matrx, m1, NULL );
32     transformmol( matdx, m1, NULL );
33     maty = newtransform( 0.,dy*(b-1),0., 0.,-phi*(b-1),0.);
34     transformmol( maty, m1, NULL );
35 
36     mergestr( m, "A", "last", m1, "sense", "first" );
37     mergestr( m, "B", "first", m1, "anti", "last" );
38     if( b > 1 ){
39         connectres( m, "A", b - 1, "O3'", b, "P" );
40         connectres( m, "B", 1, "O3'", 2, "P" );
41     }
42     ttw += TWIST; if( ttw >= 360.0 ) ttw -= 360.0;
43 }
44 putpdb( "nuc.pdb", m );
```

Finding the radius of the superhelix is a little tricky. In general a single turn of the helix will not contain an integral number of base pairs. For example, using typical numbers of 1.75 turns and 145 base pairs requires 82.9 base pairs to make one turn. An approximate solution can be found by considering the ideal superhelix that the DNA duplex is wrapped around. Let L be the arc length of this helix. Then $L\cos(\theta)$ is the arc length of its projection into the XZ plane. Since this projection is an overwound circle, L is also equal to $2\pi rt$, where t is the number of turns and r is the unknown radius. Now L is not known but is approximately $3.38(n-1)$. Substituting and solving for $r$gives Eq. 40.2.

The resulting nab code is shown in Program 2. This code requires three arguments—the number of turns, the number of base pairs and the winding angle. In lines 15-17, the helical rise (dy), twist (phi) and radius (rad) are computed according to the formulas developed above.

Two constant transformation matrices, matdx and matrx are created in lines 19-20. matdx is used to move the newly created base pair along the X-axis to the circle that is the helix's projection onto the XZ plane. matrx is used to rotate the new base pair about the X-axis so it will be tangent to the local helix of spirally wound duplex. The model of the nucleosome will be built in the molecule m which is created and given two strands "A" and "B" in line 23. The variable ttw will hold the total local helical twist for each base pair.

The molecule is created in the loop in lines 25-43. The user specified function getbase() takes the number of the current base pair (b) and returns two strings that specify the actual nucleotides to use at this position. These two strings are converted into a single base pair using the nab builtin wc_helix(). The new base pair is in the XY plane with its origin at the global origin and its helical axis along Z oriented so that the 5'-3' direction is positive.

Each base pair must be rotated about its Z-axis so that when it is added to the global helix it has the correct amount of helical twist with respect to the previous base. This rotation is performed in lines 29-30. Once the base pair has the correct helical twist it must rotated about the X-axis so that its local origin will be tangent to the global helical axes (line 31).

The properly-oriented base is next moved into place on the global helix in two stages in lines 32-34. It is first moved along the X-axis (line 32) so it intersects the circle in the XZ plane that is projection of the duplex's helical axis. Then it is simultaneously rotated about and displaced along the global Y-axis to move it to final place in the nucleosome. Since both these movements are with respect to the same axis, they can be combined into a single transformation.

The newly positioned base pair in m1 is added to the growing molecule in m using two calls to the nab builtin mergestr(). Note that since the two strands of a DNA duplex are antiparallel, the base of the "sense" strand of molecule m1 is added *after* the last base of the "A" strand of molecule m and the base of the "anti" strand of molecule m1 is *before* the first base of the "B" strand of molecule m. For all base pairs except the first one, the new base pair must be bonded to its predecessor. Finally, the total twist (ttw) is updated and adjusted to remain in the interval [0,360) in line 42. After all base pairs have been created, the loop exits, and the molecule is written out. The coordinates are saved in PDB format using the nab builtin putpdb().

40.4. Wrapping DNA Around a Path

This last code develops two nab programs that are used together to wrap B-DNA around a more general open curve specified as a cubic spline through a set of points. The first program takes the initial set of points defining the curve and interpolates them to produce a new set of points with one point at the location of each base pair. The new set of points always includes the first point of the original set but may or may not include the last point. These new points are read by the second program which actually bends the DNA.

The overall strategy used in this example is slightly different from the one used in both the circular DNA and nucleosome codes. In those codes it was possible to directly compute both the orientation and position of each base pair. This is not possible in this case. Here only the location of the base pair's origin can be computed directly. When the base pair is placed at that point its helical axis will be tangent to the curve and point in the right direction, but its rotation about this axis will be arbitrary. It will have to be rotated about its new helical axis to give the proper amount of helical twist to stack it properly on the previous base. Now if the helical twist of a base pair is determined with respect to the previous base pair, either the first base pair is left in an arbitrary orientation, or some other way must be devised to define the helical of it. Since this orientation will depend both on the curve and its ultimate use, this code leaves this task to the user with the result that the helical orientation of the first base pair is undefined.

40.4.1. Interpolating the Curve

This section describes the code that finds the base pair origins along the curve. This program takes an ordered set of points

$$p\ 1\ ,p\ 2\ ,...,pn$$

and interpolates it to produce a new set of points

$$np\ 1\ ,np\ 2\ ,...,np\ m$$

such that the distance between each np_i and np_{i+1} is constant, in this case equal to 3.38 which is the rise of an ideal B-DNA duplex. The interpolation begins by setting np_1 to p_1 and continues through the p_i until a new point np_m has been found that is within the constant distance to p_n without having gone beyond it.

The interpolation is done via spline() [45] and splint(), two routines that perform a cubic spline interpolation on a tabulated function

$$y_i = f(x_i)$$

In order for spline()/splint() to work on this problem, two things must be done. These functions work on a table of (x_i, y_i) pairs, of which we have only the y_i. However, since the only requirement imposed on the $x_i$is that they be monotonically increasing we can simply use the sequence 1 , 2 , ... , n for the x_i, producing the producing the table (i, y_i). The second difficulty is that spline()/splint() interpolate along a one dimensional curve but we need an interpolation along a three dimensional curve. This is solved by creating three different splines, one for each of the three dimensions.

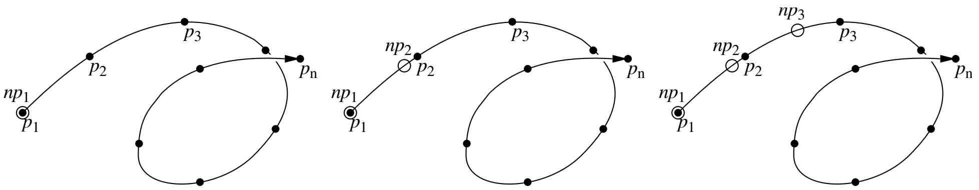
spline()/splint() perform the interpolation in two steps. The function spline() is called first with the original table and computes the value of the second derivative at each point. In order to do this, the values of the second derivative at two points must be specified. In this code these points are the first and last points of the table, and the values chosen are 0 (signified by the unlikely value of `1e30` in the calls to `spline()`). After the second derivatives have been computed, the interpolated values are computed using one or more calls to `splint()`.

What is unusual about this interpolation is that the points at which the interpolation is to be performed are unknown. Instead, these points are chosen so that the distance between each point and its successor is the constant value `RISE`, set here to 3.38 which is the rise of an ideal B-DNA duplex. Thus, we have to search for the points and most of the code is devoted to doing this search. The details follow the listing.

```
// Program 11 - Build DNA along a curve
#define RISE    3.38

#define EPS 1e-3
#define APPROX(a,b) (fabs((a)-(b))<=EPS)
#define MAXI    20

#define MAXPTS  150
int npts;
float   a[ MAXPTS ];
float   x[ MAXPTS ], y[ MAXPTS ], z[ MAXPTS ];
float   x2[ MAXPTS ], y2[ MAXPTS ], z2[ MAXPTS ];
float   tmp[ MAXPTS ];

string  line;

int i, li, ni;
float   dx, dy, dz;
float   la, lx, ly, lz, na, nx, ny, nz;
float   d, tfrac, frac;

int spline();
int splint();

for( npts = 0; line = getline( stdin ); ){
    npts = npts + 1;
    a[ npts ] = npts;
    sscanf( line, "%lf %lf %lf",
        x[ npts ], y[ npts ], z[ npts ] );
}

spline( a, x, npts, 1e30, 1e30, x2, tmp );
spline( a, y, npts, 1e30, 1e30, y2, tmp );
spline( a, z, npts, 1e30, 1e30, z2, tmp );

li = 1; la = 1.0; lx = x[1]; ly = y[1]; lz = z[1];
printf( "%8.3f %8.3f %8.3f\\n", lx, ly, lz );

while( li < npts ){
    ni = li + 1;
    na = a[ ni ];
    nx = x[ ni ]; ny = y[ ni ]; nz = z[ ni ];
    dx = nx - lx; dy = ny - ly; dz = nz - lz;
    d = sqrt( dx*dx + dy*dy + dz*dz );
    if( d > RISE ){
        tfrac = frac = .5;
        for( i = 1; i <= MAXI; i = i + 1 ){
            na = la + tfrac * ( a[ni] - la );
            splint( a, x, x2, npts, na, nx );
            splint( a, y, y2, npts, na, ny );
51             splint( a, z, z2, npts, na, nz );
52             dx = nx - lx; dy = ny - ly; dz = nz - lz;
53             d = sqrt( dx*dx + dy*dy + dz*dz );
54             frac = 0.5 * frac;
55             if( APPROX( d, RISE ) )
56                 break;
57             else if( d > RISE )
58                 tfrac = tfrac - frac;
59             else if( d < RISE )
60                 tfrac = tfrac + frac;
61         }
62         printf( "%8.3f %8.3f %8.3f\n", nx, ny, nz );
63     }else if( d < RISE ){
64         li = ni;
65         continue;
66     }else if( d == RISE ){
67         printf( "%8.3f %8.3f %8.3f\n", nx, ny, nz );
68         li = ni;
69     }
70     la = na;
71     lx = nx; ly = ny; lz = nz;
72 }
```

Execution begins in line 25 where the points are read from stdin one point or three numbers/line and stored in the three arrays x, y and z. The independent variable for each spline, stored in the array a is created at this time holding the numbers 1 to npts. The second derivatives for the three splines, one each for interpolation along the X, Y and Z directions are computed in lines 32-34. Each call to spline() has two arguments set to 1e30 which indicates that the second derivative values should be 0 at the first and last points of the table. The first point of the interpolated set is set to the first point of the original set and written to stdout in lines 36-37.

The search that finds the new points is lines 39-72. To see how it works consider the figure below. The dots marked $p_1, p_2, \ldots, p_n$ correspond to the original points that define the spline. The circles marked np_1, np_2, np_3 represent the new points at which base pairs will be placed. The curve is a function of the parameter a, which as it ranges from 1 to $npts$ sweeps out the curve from (x_1, y_1, z_1) to $(x_{npts}, y_{npts}, z_{npts})$. Since the original points will in general not be the correct distance apart we have to find new points by interpolating between the original points.



The search works by first finding a point of the original table that is at least RISE distance from the last point found. If the last point of the original table is not far enough from the last point found, the search loop exits and the program ends. However, if the search does find a point in the original table that is at least RISE distance from the last point found, it starts an interpolation loop in lines 47-61 to zero on the best value of a that will produce a new point that is the correct distance from the previous point. After this point is found, the new point becomes the last point and the loop is repeated until the original table is exhausted.

The main search loop uses li to hold the index of the point in the original table that is closest to, but does not pass, the last point found. The loop begins its search for the next point by assuming it will be before the next point in the original table (lines 40-42). It computes the distance between this point (nx,ny,nz) and the last point (lx,ly,lz) in lines 43-44 and then takes one of three actions depending it the distance is greater than RISE (lines 46-62), less than RISE (lines 64-65) or equal to RISE (lines 67-68).

If this distance is greater than RISE, then the desired point is between the last point found which is the point generated by la and the point corresponding to a[ni]. Lines 46-61 perform a bisection of the interval (la,a[ni]], a process that splits this interval in half, determines which half contains the desired point, then splits that half and continues in this fashion until the either the distance between the last and new points is close enough as determined by the macro APPROX() or MAXI subdivisions have been at made, in which case the new point is taken to be the point computed after the last subdivision. After the bisection the new point is written to stdout (line 62) and execution skips to line 70-71 where the new values na and (nx,ny,nz) become the last values la and (lx,ly,lz) and then back to the top of the loop to continue the interpolation. The macro APPROX() defined in line 4, tests to see if the absolute value of the difference between the current distance and RISE is less than EPS, defined in line 3 as 10^{-3}. This more complicated test is used instead of simply testing for equality because floating point arithmetic is inexact, which means that while it will get close to the target distance, it may never actually reach it.

If the distance between the last and candidate points is less than RISE, the desired point lies beyond the point at a[ni]. In this case the action is lines 64-65 is performed which advances the candidate point to li+2 then goes back to the top of the loop (line 38) and tests to see that this index is still in the table and if so, repeats the entire process using the point corresponding to a[li+2]. If the points are close together, this step may be taken more than once to look for the next candidate at a[li+2], a[li+3], etc. Eventually, it will find a point that is RISE beyond the last point at which case it interpolates or it runs out points, indicating that the next point lies beyond the last point in the table. If this happens, the last point found, becomes the last point of the new set and the process ends.

The last case is if the distance between the last point found and the point at a[ni] is exactly equal to RISE. If it is, the point at a[ni] becomes the new point and li is updated to ni. (lines 67-68). Then lines 70-71 are executed to update la and (lx,ly,lz) and then back to the top of the loop to continue the process.

40.4.2. Driver Code

This section describes the main routine or driver of the second program which is the actual DNA bender. This routine reads in the points, then calls putdna() (described in the next section) to place base pairs at each point. The points are either read from stdin or from the file whose name is the second command line argument. The source of the points is determined in lines 8-18, being stdin if the command line contained a single arguments or in the second argument if it was present. If the argument count was greater than two, the program prints an error message and exits. The points are read in the loop in lines 20-26. Any line with a # in column 1 is a comment and is ignored. All other lines are assumed to contain three numbers which are extracted from the string, line and stored in the point array pts by the nab builtin sscanf() (lines 23-24). The number of points is kept in npts. Once all points have been read, the loop exits and the point file is closed if it is not stdin. Finally, the points are passed to the function putdna() which will place a base pair at each point and save the coordinates and connectivity of the resulting molecule in the pair of files dna.path.pdb and dna.path.bnd.

```
1  // Program 12 - DNA bender main program
2  string      line;
3  file        pf;
4  int     npts;
5  point       pts[ 5000 ];
6  int     putdna();
7
8  if( argc == 1 )
9      pf = stdin;
10 else if( argc > 2 ){
11     fprintf( stderr, "usage: %s [ path-file ]\\n",
12         argv[ 1 ], argv[ 2 ] );
13     exit( 1 );
14 }else if( !( pf = fopen( argv[ 2 ], "r" ) ) ){
15     fprintf( stderr, "%s: can't open %s\\n",
16         argv[ 1 ], argv[ 2 ] );
17     exit( 1 );
18 }
19
20 for( npts = 0; line = getline( pf ); ){
21     if( substr( line, 1, 1 ) != "#" ){
22         npts = npts + 1;
```

```
        sscanf( line, "%lf %lf %lf",
            pts[ npts ].x, pts[ npts ].y, pts[ npts ].z );
    }
}

if( pf != stdin )
    fclose( pf );

putdna( "dna.path", pts, npts );
```

40.4.3. Wrap DNA

Every nab molecule contains a frame, a movable handle that can be used to position the molecule. A frame consists of three orthogonal unit vectors and an origin that can be placed in an arbitrary position and orientation with respect to its associated molecule. When the molecule is created its frame is initialized to the unit vectors along the global X, Y and Z axes with the origin at (0,0,0).

nab provides three operations on frames. They can be defined by atom expressions or absolute points (setframe() and setframep()), one frame can be aligned or superimposed on another (alignframe()) and a frame can be placed at a point on an axis (axis2frame()). A frame is defined by specifying its origin, two points that define its X direction and two points that define its Y direction. The Z direction is X×Y. Since it is convenient to not require the original X and Y be orthogonal, both frame creation builtins allow the user to specify which of the original X or Y directions is to be the true X or Y direction. If X is chosen then Y is recreated from Z×X; if Y is chosen then X is recreated from Y×Z.

When the frame of one molecule is aligned on the frame of another, the frame of the first molecule is transformed to superimpose it on the frame of the second. At the same time the coordinates of the first molecule are also transformed to maintain their original position and orientation with respect to their own frame. In this way frames provide a way to precisely position one molecule with respect to another. The frame of a molecule can also be positioned on an axis defined by two points. This is done by placing the frame's origin at the first point of the axis and aligning the frame's Z-axis to point from the first point of the axis to the second. After this is done, the orientation of the frame's X and Y vectors about this axis is undefined.

Frames have two other properties that need to be discussed. Although the builtin alignframe() is normally used to position two molecules by superimposing their frames, if the second molecule (represented by the second argument to alignframe()) has the special value NULL, the first molecule is positioned so that its frame is superimposed on the global X, Y and Z axes with its origin at (0,0,0). The second property is that when nab applies a transformation to a molecule (or just a subset of its atoms), only the atomic coordinates are transformed. The frame's origin and its orthogonal unit vectors remain untouched. While this may at first glance seem odd, it makes possible the following three stage process of setting the molecule's frame, aligning that frame on the *global* frame, then transforming the molecule with respect to the global axes and origin which provides a convenient way to position and orient a molecule's frame at arbitrary points in space. With all this in mind, here is the source to putdna() which bends a B-DNA duplex about an open space curve.

```
// Program 13 - place base pairs on a curve.
point       s_ax[ 4 ];
int     getbase();

int putdna( string mname, point pts[ 1 ], int npts )
{
    int p;
    float   tw;
    residue r;
    molecule    m, m_path, m_ax, m_bp;
    point   p1, p2, p3, p4;
    string  sbase, abase;
    string  aex;
    matrix  mat;

    m_ax = newmolecule();
    addstrand( m_ax, "A" );
    r = getresidue( "AXS", "axes.rlb" );
    addresidue( m_ax, "A", r );
    setxyz_from_mol( m_ax, NULL, s_ax );

    m_path = newmolecule();
    addstrand( m_path, "A" );

    m = newmolecule();
    addstrand( m, "A" );
    addstrand( m, "B" );

    for( p = 1; p < npts; p = p + 1 ){
        setmol_from_xyz( m_ax, NULL, s_ax );
        setframe( 1, m_ax,
            "::ORG", "::ORG", "::SXT", "::ORG", "::CYT" );
        axis2frame( m_path, pts[ p ], pts[ p + 1 ] );
        alignframe( m_ax, m_path );
        mergestr( m_path, "A", "last", m_ax, "A", "first" );
        if( p > 1 ){
            setpoint( m_path, sprintf( "A:%d:CYT",p-1 ), p1 );
            setpoint( m_path, sprintf( "A:%d:ORG",p-1 ), p2 );
            setpoint( m_path, sprintf( "A:%d:ORG",p ), p3 );
            setpoint( m_path, sprintf( "A:%d:CYT",p ), p4 );
            tw = 36.0 - torsionp( p1, p2, p3, p4 );
            mat = rot4p( p2, p3, tw );
            aex = sprintf( ":%d:", p );
            transformmol( mat, m_path, aex );
            setpoint( m_path, sprintf( "A:%d:ORG",p ), p1 );
            setpoint( m_path, sprintf( "A:%d:SXT",p ), p2 );
            setpoint( m_path, sprintf( "A:%d:CYT",p ), p3 );
            setframep( 1, m_path, p1, p1, p2, p1, p3 );
        }

        getbase( p, sbase, abase );
        m_bp = wc_helix( sbase, "", "dna",
            abase, "", "dna",
            2.25, -5.0, 0.0, 0.0 );
        alignframe( m_bp, m_path );
        mergestr( m, "A", "last", m_bp, "sense", "first" );
        mergestr( m, "B", "first", m_bp, "anti", "last" );
        if( p >  1 ){
            connectres( m, "A", p - 1, "O3'", p, "P" );
            connectres( m, "B", 1, "P", 1, "O3'" );
        }
    }

    putpdb( mname + ".pdb", m );
    putbnd( mname + ".bnd", m );
};
```

putdna() takes three arguments—name, a string that will be used to name the PDB and bond files that hold the bent duplex, pts an array of points containing the origin of each base pair and npts the number of points in the array. putdna() uses four molecules. m_ax holds a small artificial molecule containing four atoms that is a proxy for the some of the frame's used placing the base pairs. The molecule m_path will eventually hold one copy of m_ax for each point in the input array. The molecule m_bp holds each base pair after it is created by wc_helix() and m will eventually hold the bent dna. Once again the function getbase() (to be defined by the user) provides the mapping between the current point (p) and the nucleotides required in the base pair at that point.

Execution of putdna() begins in line 16 with the creation of m_ax. This molecule is given one strand "A", into which is added one copy of the special residue AXS from the standard nab residue library "axes.rlb" (lines 17-19). This residue contains four atoms named ORG, SXT, CYT and NZT. These atoms are placed so that ORG is at (0,0,0) and SXT, CYT and NZT are 1o along the X, Y and Z axes respectively. Thus the residue AXS has the exact geometry as the molecules initial frame—three unit vectors along the standard axes centered on the origin. The initial coordinates of m_ax are saved in the point array s_ax. The molecules m_path and m are created in lines 22-23 and 25-27 respectively.

The actual DNA bending occurs in the loop in lines 29-62. Each base pair is added in a two stage process that uses m_ax to properly orient the frame of m_path, so that when the frame of new the base pair in m_bp is aligned on the frame of m_path, the new base pair will be correctly positioned on the curve.

Setting up the frame is done is lines 30-49. The process begins by restoring the original coordinates of m_ax (line 30), so that the the atom ORG is at (0,0,0) and SXT, CYT and NZT are each 1o along the global X, Y and Z axes. These atoms are then used to redefine the frame of m_ax (line 32-33) so that it is equal to the three standard unit vectors at the global origin. Next the frame of m_path is aligned so that its origin is at pts[p] and its Z-axis points from pts[p] to pts[p+1] (line 34). The call to alignframe() in line 34 transforms m_ax to align its frame on the frame of m_path, which has the effect of moving m_ax so that the atom ORG is at pts[p] and the ORG—NZT vector points towards pts[p+1]. A copy of the newly positioned m_ax is merged into m_path in line 35. The result of this process is that each time around the loop, m_path gets a new residue that resembles a coordinate frame located at the point the new base pair is to be added.

When nab sets a frame from an axis, the orientation of its X and Y vectors is arbitrary. While this does not matter for the first base pair for which any orientation is acceptable, it does matter for the second and subsequent base pairs which must be rotated about their Z axis so that they have the proper helical twist with respect to the previous base pair. This rotation is done by the code in lines 37-48. It does this by considering the torsion angle formed by the fours atoms—CYT and ORG of the previous AXS residue and ORG and CYT of the current AXS residue. The coordinates of these points are determined in lines 37-40. Since this torsion angle is a marker for the helical twist between pairs of the bent duplex, it must be 36.0o. The amount of rotation required to give it the correct twist is computed in line 41. A transformation matrix that will rotate the new AXS residue about the ORG—ORG axis by this amount is created in line 42, the atom expression that names the AXS residue is created in line 43 and the residue rotated in line 44. Once the new residue is given the correct twist the frame m_path is moved to the new residue in lines 45-48.

The base pair is added in lines 51-60. The user defined function getbase() converts the point number (p) into the names of the nucleotides needed for this base pair which is created by the nab builtin wc_helix(). It is then placed on the curve in the correct orientation by aligning its frame on the frame of m_path that we have just created (line 55). The new pair is merged into m and bonded with the previous base pair if it exists. After the loop exits, the bend DNA duplex coordinates are saved as a PDB file and the connectivity as a bnd file in the calls to putpdb() and putbnd() in lines 64-65, whereupon putdna() returns to the caller.

40.5. Other examples

There are several additional pedagogical (and useful!) examples in *$AMBERHOME/examples*. These can be consulted to gain ideas of how some useful molecular manipulation programs can be constructed.

- The *peptides* example was created by Paul Beroza to construct peptides with given backbone torsion angles. The idea is to call linkprot to create a peptide in an extended conformation, then to set frames and do rotations to construct the proper torsions. This can be used as just a stand-alone program to perform this task, or as a source for ideas for constructing similar functionality in other nab programs.
- The *suppose* example was created by Jarrod Smith to provide a driver to carry out rms-superpositions. It has a man page that shows how to use it.

41. amberlite: Some AmberTools-Based Utilities

Romain M. Wolf
Novartis Institutes for Biomedical Research, NIBR, Basle, Switzerland

AMBER "Lite" is a small set of utilities making use of the free AMBER Tools package (currently for version 1.4 or higher). The main focus is on the preparation of files for MM(GB)(PB)/SA-type simulations. The utilities can be used as delivered or they can serve as a starting point for further development. Examples are included to illustrate the concepts or to test the correct functioning of the installation. The text also contains a (very) condensed introduction to some AMBER file preparation concepts.

The AMBER Lite package (© Novartis Institutes for Biomedical Research, Basel, Switzerland) is free software under the GNU General Public License (GPL), as are the parts on which the package builds, namely the Amber tools *ptraj*, *leap*, *antechamber*, *sqm*, *pbsa* and the *NAB package*.

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

41.1. Introduction

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

One section (Appendix 12.1) is dedicated to the preparation of PDB files prior to use them with AMBER. In my own experience, this is a critical part in setting up simulations. Scanning through the AMBER Mail Reflector, I find many reported problems and questions which originate from "bad" or badly prepared PDB starting files.

Another section (Appendix 19) gives a brief introduction on AMBER "masks" and NAB "atom expressions", used to select parts of molecular structures. Users should also read in detail the corresponding information in original AMBER documentations. Wrong partial selections are tricky because they may often go unnoticed, i.e., everything seems to run OK but the results are totally flawed.

41.1.1. Python Scripts

The following Python scripts are currently included:

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

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

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

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

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

41.1.2. NAB Applications

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

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

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

The following NAB-based tools are currently included:

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

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

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

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

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

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

41.2. Coordinates and Parameter-Topology Files

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

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

The current tool delivered with AMBER to prepare coordinate (CRD or PDB) and parameter-topology (PRM) files is called ***leap*** (*tleap* for the terminal variant and *xleap* for the graphics variant).

Since *leap* is not particularly user-friendly, a Python script ***pytleap*** (see section 41.3) has been prepared which runs the terminal version of *leap* in the background and does not require a direct interaction with *leap* itself, at least for simple tasks like preparing a protein or a receptor-ligand complex for simulations with implicit solvent.

For small organic molecules, *pytleap* first invokes ***antechamber*** [664] before passing them through *leap*, allowing the usage of the *gaff* force field[326] for organics without directly interfering with *antechamber* itself.

Chapter12 gives a short outline of the most important preparation steps required on the raw data (mostly PDB files) before using any AMBER-related tools. Those recipes may not be the most elegant ones but they work in most cases and help avoid common problems.

41.3. *pytleap*: Creating Coordinates and Parameter-Topology Files

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

Proteins (or peptides) are read as PDB files in *pytleap*. Other formats are not supported. Be sure to have a "clean" PDB file as described in Chapter 12.

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

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

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

Note, December 2016: The latest version of pytleap now uses by default ff14SB for proteins/peptides and gaff2 for ligands. This may change slightly the results shown in the following example sections. To get the same numbers (*e.g.* for test reasons), you can still explicitly specify ff03.r1 and gaff for protein and ligand FFs

41.3.1. Running *pytleap*

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

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

```
-------------------------------------------
 pytleap version 1.4 (December 2016)
-------------------------------------------
Usage: pytleap [options]

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

The command line options are presented here below:

--prot *`filename`* uses the PDB file *filename* as the protein structure. The PDB file must be "clean", according to the rules outlined in the Appendix 12.1. The *leap* module adds hydrogens with correct names (and also missing heavy atoms, if any), attributes the correct partial charges and AMBER atom types,[1] and eventually writes out the files for the protein as mentioned in section 41.3.2.

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

--lig *`filename`* uses the SDF file *filename* as the ligand structure. The ligand file **must include all hydrogens**. The structure is processed through *antechamber* that generates various files required by *tleap* to build the PRM file for the ligand. Inside *antechamber*, the ligand becomes a molecule (residue) with the name "**LIG**". This name is then taken over by *leap* and appears as such in the resulting PRM and PDB files. The name "LIG" is the default name for a ligand in the *pymdpbsa* (section 41.7). See also option `--chrg` when using the `--lig` option. **Note:** We assume here that a ligand is a **single-residue** low-molecular-weight organic molecule.

--cplx *`filename`* **(no extension!)** will generate the AMBER files PRM, CRD and a PDB file of the complex of the protein and the ligand read in with the `--prot` and `--lig` options. When generating AMBER files for the complex, the files for the individual protein and ligand are always generated also. They are useful when running MM(GB)(PB)/SA computations later (section 41.7). **This option only makes sense when both the `--prot` and `--lig` options are also chosen**.

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

--chrg *`integer`* must be used if an organic ligand read from an SDF file is formally charged (even if the charge is also given in the SDF file). For neutral ligands, this option can be omitted. For charged ligands however, it is required! Enter it as an integer reflecting the correct total charge of the ligand. The computation of partial charges via the AM1-BCC method[329, 330] will fail if the formal charge on the ligand does not make sense with the chemical structure including all hydrogens and *pytleap* will quit.

--rad *`radius_type`* is used to choose the atomic radii for Generalized-Born. The default radius type is the "*modified Bondi*" option to be used with the GB option `gb` set to 1. For `gb` = 2 or 5, Table 4.1 suggests the radius type `mbondi2`. For `gb` = 7, use the radius type `bondi` and for `gb` = 8, use the radius type `mbondi`3.

--disul and **--sspep** *`filename`* are used to generate disulfide bonds. Disulfide bridges must be prepared in the original PDB file by renaming the involved cysteine residues from CYS to CYX (see 12.2). The *filename* in this option must relate to a file that contains pairwise integer numbers of cysteine residue names to be connected (one pair per line!). These numbers must correspond to the ones in the original PDB file![2] See the example in section 41.8.1. We consider that this explicit formation of disulfide bonds is to be preferred over "automatic" S-S bond formation, be it by using Sγ distances or by relying on `CONECT` records in PDB files. **NOTE: `--disul` applies to the file read in via `--prot` while `--sspep` is applied to the molecule read in via –pep.** If both proteins (peptides) have disulfide bonds, you must use separate definition files for the respective S-S links!

--sspep: see above.

---

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

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

--pfrc *filename* specifies the force field parameter set for the protein. Since AMBER can use different force fields, this option allows to choose among them. The selection actually does not call the parameter file itself but a *leap* command file that initializes it. These special *leap* files all have a name *leaprc.xxxx* and are retrieved when the `AMBERHOME` environment variable is set correctly. **You must only specify the *xxxx* part of the name!** Thus, `ff99` selects the parm99 parameter set, while the **default** `ff03.r1` selects the latest parm03 force field with the correct charges for N- and C-terminal residues also. Make sure to have this file (with the full name *leaprc.ff03.r1* included in the directory where all default `leap` command files are kept.[3]

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

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

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

41.3.2. Output from *pytleap*

Output from *pytleap* varies with the chosen command line options (see 41.3.1). Coordinate (CRD) files, parameter-topology (PRM) files and a corresponding PDB file are always generated. Hydrogen names in the output PDB files are "wrapped", making these files readable also by elder software packages which require this format. Note that the actual atom names in the PRM file are unwrapped. This has no consequence on computations. However, special residue names like HIE, HID, HIP, CYX, etc., are kept and may lead to flawed representations of the PDB files in software packages which do not recognize these residue names. The *ambpdb* routine included with AMBER Tools can be used to regenerate "standard" residue names if you need them.

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

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

A special SYBYL MOL2 file is created when running *pytleap* on a ligand (i.e., a low-molecular-weight organic compound which is processed through *antechamber*). This file has the format of a generic MOL2 file, apart from the fact that atom types are not SYBYL but *gaff* atom types. The name of this file is **`filename.ac.mol2`**, with `.ac.` marking it as a file generated by *antechamber*.[4] The partial charges are those from the AM1-BCC method.[329, 330]

Some additional files may be left over when *antechamber* is invoked. One **important file** is the **`*.leap.frcmod`** file containing additional parameters which are not in the original *gaff* parameter file. They are generated based on equivalences, "guessing", or empirical rules described the gaff paper.[326] The `frcmod` file can also be used as a quality check for the ligand parameters. Large *frcmod* files with many "guessed" parameters (especially for torsion angles) should be considered carefully.

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

---

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

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

41.3.3. Error Checking

If you have experience with the *leap* application, look at the *leap.cmd* file that was created via *pytleap*. All the options that you have chosen should be represented as correct *leap* command lines. Furthermore, the *leap.output* and *leap.log* files should not show any errors, at best some warnings. If in doubt that the parameter-topology files have been correctly generated, look at these warnings and decide if they are benign. Eventually, the NAB application *ffgbsa* described below (section 41.4) can be used to run a single AMBER energy evaluation on the system. If the results returned by *ffgbsa* look very strange for a supposedly reasonable structure, you probably have a serious issue with your set of CRD, PDB, and PRM files.

41.4. Energy Checking Tool: *ffgbsa*

The NAB routine *ffgbsa* is an energy function called by the *pymdpbsa* application presented later (section 41.7). It can also be used as a standalone routine to check the AMBER energy of a molecular system and to test the correct working of a PDB/PRM file combination. It is invoked as:

```
ffgbsa pdb prm gbflag saflag
```

The order in the command line input is compulsory! `pdb` is the PDB file of the system and `prm` the related PRM file. `gbflag` is a flag to switch on one of the Generalized-Born (GB) options in AMBER and can be 1, 2, 5, 7 or 8. These values correspond to the "*igb=*" options in AMBER commands and stand for different implementations of the GB scheme. Other values switch off GB and a simple distance-dependent dielectric function $\varepsilon = r_{ij}$ is used. Note that different GB or PB selections may require a different choice for atom radii via LeaP or ParmEd; see Table 4.1.

When `saflag` = 1, the solvent-accessible surface area (SASA) is also computed (via the *molsurf* routine included with NAB) and returned in Å^2, together with a SASA energy term which is simply $SASA * 0.0072$ in this case. The default cutoff for non-bonded interactions is 100 Å , i.e., virtually no cutoff for most systems. An example for the usage of *ffgbsa* is given in section 41.8.2.

Remarks regarding the usage of *molsurf*:
The correct way to evaluate the SASA is to augment the radii of all atoms by the probe radius (usually 1.4 Å) and then run *molsurf* with a probe of radius of zero. This is also the implementation in *ffgbsa* here. The atom radii values are given in Table 41.1.

Table 41.1.: *Atom Radii Used in* molsurf

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

Note: In some rare cases *molsurf* fails to give back a valid surface area. Scripts calling *ffgbsa* must be prepared to capture this. The *pymdpbsa* procedure described later catches such instances and excludes value sets in which the error occurs from the statistical analysis (cf. end of section 41.7.5.1).

41.5. Energy Minimizer: *minab*

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

The NAB routine *minab* uses the conjugate gradient minimizer of NAB to refine the energy of a system. To circumvent cutoff problems,[5] the cutoff for non-bonded interactions (vdW and Coulomb) is set to 100 Å and that for GB is fixed to 15 Å. The non-bonded list is not updated at all. The default for the gradient rms is set to 0.1.

For large systems, this is far from efficient. However, as stated above, the main purpose of this routine is to get rid of hotspots prior to running MD and in general, a few hundred iterations are sufficient to guarantee a decent structure for MD, especially when the MD starts with a heat-up phase as used in the *mdnab* application described in section 41.6.

The *minab* routine is invoked by:

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

Just typing `minab` without arguments gives a help screen. The explanation for the arguments follows:

- `pdb` and `prm` are the PDB and **corresponding** PRM file of the system;
- `pdbout` becomes the PDB file of the refined system;
- `gbflag` is the GB flag which can be 1, 2, 5, 7, or 8 while any other value switches to distance-dependent dielectrics (as in section 41.4);
- `niter` is the maximum number of iterations;

and for the optional arguments:

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

The `restraints` entry must be an atom expression according to the NAB rules outlined in 19.2. If for example all Cα atoms should be restrained, this entry would be '::CA'. **If the restraint mask is given, the restraint potential `resforce` must also be specified.**

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

41.6. Molecular Dynamics "Lite": *mdnab*

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

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

The following (non-mutable) defaults are used:

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

---

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

A heating and equilibration phase is automatically invoked prior to the actual production trajectory recording: 100 steps from 50 to 100 K, 300 steps from 100 to 150 K, 600 steps from 150 to 200 K, 1000 steps from 200 to 250 K, 3000 steps from 250 to 300 K, and an additional 10000 steps at 300 K.[6]

mdnab is started by

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

The command *mdnab* without arguments lists the possible arguments, the **sequence** of which is **compulsory**. The command line arguments are similar to those in *minab*:

- `pdb` and `prm` are the PDB and corresponding PRM file of the system;
- `traj` is the name for the production phase trajectory which will be saved in the binary "binpos" format (the extension `.binpos` is automatically attached);[7]
- `gbflag` is the GB flag which can be 1, 2, 5, 7, or 8 (as in section 41.4), or anything else to switch off GB and use a distance-dependent dielectric function $\varepsilon = r_{ij}$;
- `picosecs` is the total number of picoseconds to run the production phase;

with the optional arguments:

- `restraints` specifies atoms to be tethered in their motion (given as a NAB atom expression **between quotes**, see section 19.2);
- `resforce` is the restraint potential in kcal·mol^{-1}·Å^{-2} **which has to be given** if a restraint expression is specified.

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

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

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

41.7.1. Brief Overview on MM(GB)(PB)/SA Concepts

The original MM(GB)(PB)/SA procedure was developed in the late 1990's and the user should refer to some original papers on this subject.[590, 591, 665, 666] The goal was to develop a relatively fast molecular-mechanics (-dynamics) based method to evaluate free energies of interactions. MM stands for Molecular Mechanics, PB for Poisson-Boltzmann, and SA for Surface Area. You may also wish to refer to reviews summarizing many of the applications of this model,[591, 592] as well as to papers describing some of its applications.[593–597]

The free energy for each species (ligand, receptor, or complex) is decomposed into a gas-phase energy ("enthalpy"), a solvation free energy and an entropy term, as shown in equation 41.1.

$$G = E_{gas} + G_{solv} - T \cdot S \tag{41.1}$$

$$= E_{bat} + E_{vdW} + E_{coul} + G_{solv,polar} + G_{solv,nonpolar} - T \cdot S \tag{41.2}$$

---

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

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

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

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

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

$$G_{solv,nonpolar} = \gamma \cdot SASA + const. \tag{41.3}$$

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

$$G_{solv,nonpolar} = p \cdot SAV + const. \tag{41.4}$$

with p having the dimensions of pressure.

In a more sophisticated approach, $G_{solv,nonpolar}$ can be spThese values correspond to the "*igb=*" options in AMBER commands and stand for different implementations of the GB scheme.lit into a repulsive ("cavity") and an attractive ("dispersion") term, as described in detail in the 2007 paper of Ray Luo and coworkers.[173]

The vibrational entropy can be evaluated, for example, via normal mode analysis. It has become common practice in recent work to exclude the entropy terms from MM(GB)(PB)/SA computations. This is acceptable when only relative free energies are computed to compare similar ligands in similar receptors. Furthermore, the entropy computation is the fuzziest part of the procedure and contributes to the largest fluctuations in the overall free energy when evaluating it over a number of MD frames.

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

$$\Delta G_{int} = G_{complex} - G_{receptor} - G_{ligand} \tag{41.5}$$

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

$$\Delta G_{int} = \langle G_{complex} \rangle_{traj} - \langle G_{receptor} \rangle_{traj} - \langle G_{ligand} \rangle_{traj} \tag{41.6}$$

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

In the meantime, the method has been implemented and used in many variants, all of which have their advantages and disadvantages. The method presented hereafter is among the simplest and cheapest in terms of CPU power. It is based on a single trajectory of the complex alone. Each recorded frame is then split into receptor and ligand and equation 41.5 is applied to compute the interaction energy of the frame. The final interaction energy is then the average over the ΔG values of the selected frames. Also, the entropy is not evaluated at all.

41.7.2. Pitfalls and Error Sources

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

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

In the "one-trajectory" approach implemented here, there is an additional pitfall. Since both the receptor and the ligand are only considered in the bound state, strain energy from distortions in the complex is not evaluated.

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

41.7.3. Some Technical Remarks on *pymdpbsa*

pymdpbsa uses *ffgbsa* (see section 41.4) or the stand-alone Poisson-Boltzmann solver *pbsa* to evaluate energies. The tool *ptraj* is called to decompose the MD trajectory into individual frames for the complex, the ligand, and the receptor.

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

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

- a molecular dynamics trajectory file of the complex (any format that can be read by *ptraj*, including Z-compressed ones and binary binpos files like those created by *mdnab*, see section 41.6);
- three AMBER parameter-topology PRM files, one for the complex, one for the ligand alone, and one for the receptor alone (as created by *pytleap*, see section 41.3);

41.7.4. Running *pymdpbsa*

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

```
---------------------------------------------------------
 pymdpbsa version 0.7 (May 2011)
---------------------------------------------------------
Usage: pymdpbsa [options]

Options:
  -h, --help     show this help message and exit
  --proj=NAME    global project name
  --traj=FILE    MD trajectory file              (default: traj.binpos)
  --cprm=FILE    complex prmtop file             (default: com.prm)
  --lprm=FILE    ligand only prmtop file         (default: lig.prm)
  --rprm=FILE    receptor only prmtop file       (default: rec.prm)
  --lig=STRING   residue name of ligand          (default: LIG)
  --start=INT    first MD frame to be used       (default: 1)
  --stop=INT     last MD frame to be used        (default: 1)
  --step=INT     use every [step] MD frame       (default: 1)
  --solv=INT     0 for no solvation term (eps=r)
                 1, 2,5, 7 or 8 for GBSA
                 3 for PBSA
                 4 for PBSA/dispersion           (default: 1)
  --clean        clean up temporary files        (default: no clean)
```

You only need to specify options that are different from the default. Thus, you can avoid entering a lot of options by simply selecting file names like *com.prm*, *rec.prm*, and *lig.prm* for the PRM files, calling the trajectory file *traj.binpos*, and by giving the ligand the residue name LIG in your original structure file (the default if *pytleap* in section 41.3 was used).

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

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

`--cprm`, `--lprm`, `--rprm` are used to feed in the names for the PRM files of the complex, the ligand, and the empty receptor. None of these PRM files is generated by *pymdpbsa*. They must be specified by the user. If the *pytleap* utility (see section 41.3) has been used on a complex, these three files should have been created. If you want to use default names, rename these files to `com.prm`, `rec.prm`, and `lig.prm`.

`--lig` is used to specify the name of the ligand. This is the (up to 4 characters long) "residue" name the ligand would have in a PDB file. If the complex has been prepared via *pytleap*, the ligand name will probably be LIG (i.e., the default). Note that the ligand is supposed to be one single residue in that case. Alternatively, the ligand can also be specified by its residue number. Thus if the ligand is residue 281 in the PDB file of the complex, you may specify `--lig 281`. This also allows to have multi-residue ligands like in protein-peptide (protein-protein) complexes. If e.g. the ligand covers residues 134 to 156 in the **overall** PDB file of the complex, you can specify `--lig '134-156'`.[8]

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

`--solv` followed by an integer chooses the solvation option. The **default** is `'--solv 1'`. For values other than 1, 2, 3, 5, 7, and 8, the returned electrostatic energy term is evaluated with a distant-dependent dielectric function $\varepsilon = r_{ij}$ with no additional polar solvation correction. For values 1, 2, 5, 7, or 8, the corresponding GB variant (*igb* in AMBER) is used with a nonpolar contribution of 0.0072 * SASA (where the solvent-accessible surface SASA is computed via *molsurf*); for `solv = 3`, GB is replaced by PB and the non-polar solvation energy term is 0.005 * SASA + 0.86; for `solv = 4`, the polar solvation free energy part is computed with PB, the nonpolar portion is evaluated by a "cavity" term and a "dispersion term";[173] the detailed settings for this approach are identical to those suggested in the original *pbsa* documentation; **note that the `'--solv 4'` option is experimental at this stage and not widely tested, ...use with care**.

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

41.7.5. Details on Internal Workings and Output of *pymdpbsa*

The internal workings and the output of *pymdpbsa* vary depending on the `--solv` options. In all cases, the *ptraj* tool is called to split the trajectory into individual frames. Since each interaction energy evaluation requires three files (complex, receptor, ligand), the splitting of a trajectory with N frames results in 3·N files.[9]

41.7.5.1. Distance-Dependent Dielectrics or Generalized Born

For `--solv` = 0, 1, 2,5,7, or 8, the *ffgbsa* routine is called to evaluate energy terms. **Note**: options 7 and 8 are new ones (so use with care) and **require the bondi radii (set with the –rad option in pytleap)**. Since *ffgbsa* requires PDB files as coordinate input, the trajectory is split into individual PDB files. These files are named according to the project, the part of the structure (C for whole complex, R for receptor alone, L for ligand alone), and the frame number.

---

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

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

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

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

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

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

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

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

For `--solv` = 0, 1, 2, 5, 7, or 8, the solvent-accessible surface is computed via the NAB subroutine *molsurf* in *ffgbsa*. The surface returned by *ffgbsa* is multiplied by a surface tension of **0.0072** to yield the "nonpolar" free energy component in kcal/mol. For details about the calls to *molsurf*, see section 41.4.

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

41.7.5.2. Poisson-Boltzmann

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

Since *pbsa* requires CRD files, the trajectory is split into AMBER restart files rather then PDB files. The name giving is the same as for the PDB files (see 41.7.5.1) except that the "pdb" part in filenames is changed to "crd".

The script eventually calls *pbsa* by:

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

The generated output tables are named as for the non-PB settings in section 41.7.5.1. However, the content of the tables varies slightly:

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

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

The final evaluation summary is stored in a file with the project name and the extension "`.sum`". This file is similar to that shown in section 41.7.5.1 except that some specific terms vary. An example is shown here:

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

```
========================================================================
```

41.7.6. Using *pymdpbsa* for Single-Point Interaction Energy

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

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

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

41.7.7. Remark Concerning Poisson-Boltzmann Options `--solv 3` and `--solv 4`

The PB option has more adjustable parameters than the GB variants. The `--solv 3` option uses the following settings for calling *pbsa*:

```
&cntrl
ntx=1, imin=1, igb=10, inp=1,
/
&pb
epsout=80.0, epsin=1.0, space=0.5, bcopt=6, dprob=1.4,
cutnb=0, eneopt=2,
accept=0.001, sprob=1.6, radiopt=0, fillratio=4,
maxitn=1000, arcres=0.0625,
cavity_surften=0.005, cavity_offset=0.86
/
```

This command sequence is generated in *pymdpbsa* in the function `pbsacontrol_solv3`. Users who want to try different settings can change this code section of *pymdpbsa* to their gusto. Read the original *pbsa* documentation before doing so, however.

The `--solv 4` option to compute interaction energies is highly experimental. Users should read the original paper of Ray Luo et coworkers[173] before using this option. The command sequence generated by *pymdpbsa* is found in the source code in the function `pbsacontrol_solv4`:

```
&cntrl
ntx=1, imin=1, igb=10, inp=2
/
&pb
npbverb=0, istrng=0.0, epsout=80.0, epsin=1.0,
radiopt=1, dprob=1.6,
space=0.5, nbuffer=0, fillratio=4.0,
accept=0.001, arcres=0.25,
cutnb=0, eneopt=2,
decompopt=2, use_rmin=1, sprob=0.557, vprob=1.300,
rhow_effect=1.129, use_sav=1,
cavity_surften=0.0378, cavity_offset=-0.5692
/
```

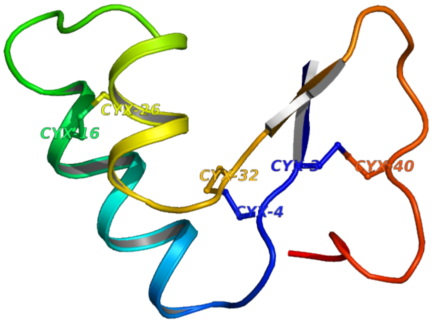


Figure 41.1.: *The 3 S-S links in* 1crn.pdb

41.8. Examples and Test Cases

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

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

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

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

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

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

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

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

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

41.8.2. Example 2: Energy Minimization of the Crambin Structure

41.8.2.1. Starting Energy

We can use the files *1crnx.leap.prm* and *1crnx.leap.pdb* which were created in section 41.8.1 to evaluate the AMBER energy terms in the unrefined crambin structure with the Generalized-Born option 1 and the SASA (nonpolar) energy:

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

The result is:

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

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

In this output, the line starting with "`ff:`" lists the total energy of the system and the components (`bad` = bond-angle-dihedral combined energy, i.e., the sum of the bonded terms). The line starting with "`sasa:`" gives the solvent-accessible surface in $\mathring{A}^2$. The final line is the result from SASA multiplied by a surface tension of 0.0072. All energies are in kcal·mol^{-1}.[10]

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

41.8.2.2. Energy Minimization with *minab*

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

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

We use the GB option 1 and request a maximum of 1000 steps. No restraints are applied. The refined coordinates go to the PDB file *crambin.min.pdb*. The output of *minab* is redirected to the text file `crambin.min.out`. The final '`&`' puts the process into the background. The minimization can be followed interactively by the command `tail -f crambin.min.out`

The last lines of the output are:

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

Figure 41.2 shows the initial (green) and refined (orange) structure of crambin. The disulfide bonds in the refined structure are shown in CPK mode to emphasize that the S-S links have been correctly assigned. If this were not the case, the respective sulfur atoms would drift apart because of non-bonded interactions.

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

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

In the `...amberlite/examples/P38` directory, the PDB file *1OUK.pdb* (P38 MAP kinase with inhibitor) is included in its original version. The structure (see Figure 41.3) contains a ligand (red), a sulfate ion (yellow),

---

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

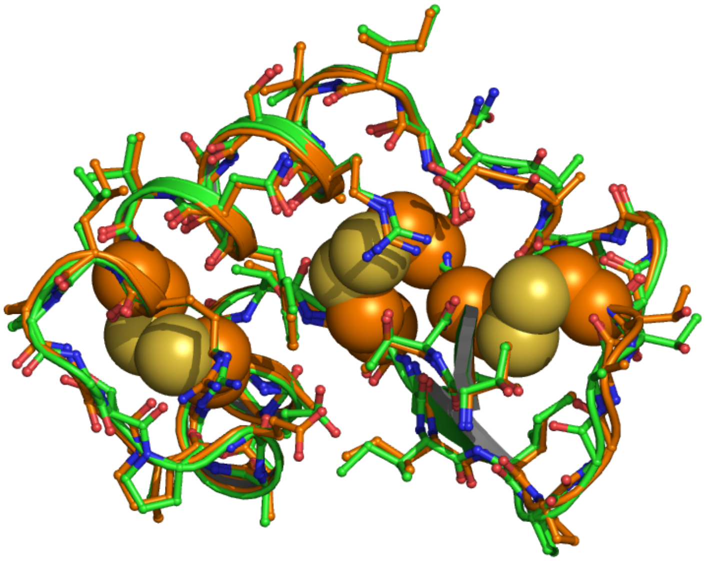


Figure 41.2.: *Starting (green) and refined (orange) coordinates of 1CRN. Disulfide bonds in the refined structure are shown in CPK mode.*

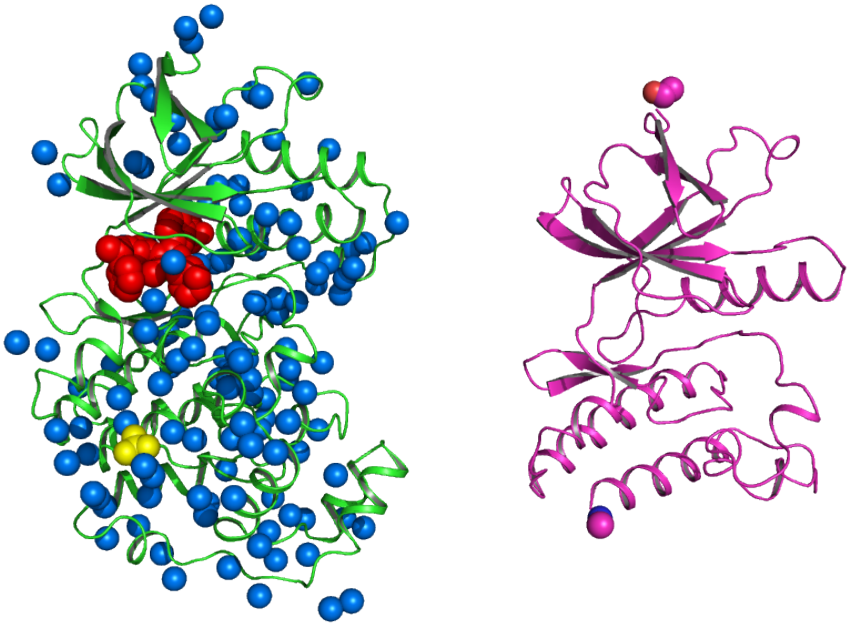


Figure 41.3.: *Left side: original structure* 1OUK.pdb *with ligand (red), sulfate (yellow) and water molecules (blue); right side: final structure* p38.pdb *with the N- and C-terminal caps.*

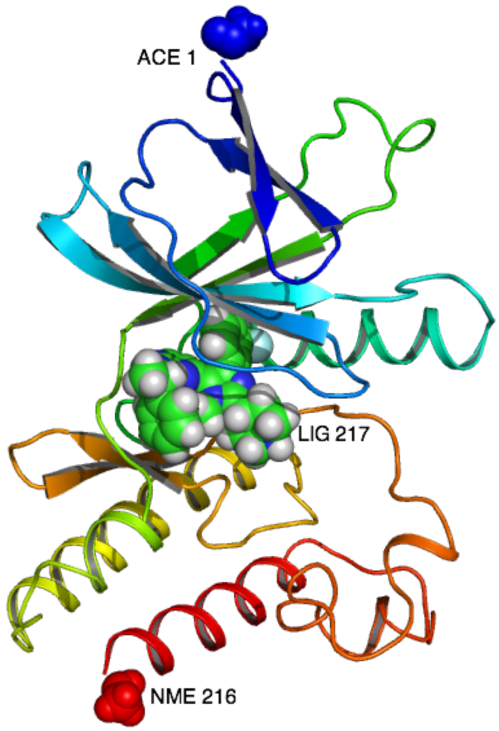


Figure 41.4.: *The final complex structure* com.leap.pdb*: The ACE cap becomes residue 1, the NME cap is residue 216, and the ligand is residue LIG 217*

and a number of water molecules (blue). The file *p38.pdb* (also included in `...amberlite/examples/P38`) was created from this PDB file by cutting off a large part of the protein and deleting everything except the heavy atoms. The resulting "nonnatural" N- and C-terminal were then completed by ACE and NME caps, respectively.[11] The resulting PDB file is "clean" for *leap* and passes without errors. The ligand was processed separately into SDF format (file `lig.sdf` in `...amberlite/examples/P38`) including all hydrogens and bond orders. This file is ready to be processed via *antechamber* before re-complexing it with the protein (see later).

41.8.3.2. Generating AMBER Files for a P38/Ligand Complex

We re-use as a receptor the reduced and corrected PDB file from section 41.8.3.1, *p38.pdb*. For the ligand, we use the *lig.sdf* file, containing the ligand with its heavy-atoms coordinates from the original pdb file *1OUK.pdb* and hydrogen atoms added via any other software that can handle this kind of problem. Note that the ligand has a formal charge of +1.

The following command line will create the PRM, CRD, and PDB files for the empty receptor, the ligand, and the complex; partial charges on the ligand will be computed via the AM1-BCC method;[329, 330] the complex will be named *com*:

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

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

The resulting AMBER files are called `*.leap.prm`, `*.leap.crd`, and `*.leap.pdb`, respectively. One set of files is generated for the ligand (`lig`), the receptor (`p38`), and the complex (`com`).[12]

---

[11] This can be done by any modeling software which allows building, but make sure that the final atom and residue names in the cap residues are those described in section 12.2.

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

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

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

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

We use the files generated in example 3 (section 41.8.3.2). In order to make use of the default settings in the command line options, we copy the respective files to the default names proposed by *pymdpbsa*: Make copies (or symbolic links) of *com.leap.prm*, *p38.leap.prm*, and *lig.leap.prm* to *com.prm*, *rec.prm*, and *lig.prm*. Also, make a copy (or symbolic link) of *com.leap.pdb* to *com.pdb*.

Now the command

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

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

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

The output of interest is the summary file `RAWPDB.sum` (see also 41.7.5.1 and 41.7.5.2):

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

---

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

```
======================================================================
```

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

We minimize the P38/ligand complex prepared in section 41.8.3. We use the (renamed) PDB file *com.pdb*, the corresponding PRM file *com.prm*, gb = 1 and a maximum of 500 iterations. We tether Cα atoms with a force constant of 1.0 kcal·mol^{-1}·Å^{-2}. The refined coordinates are written to com.min.pdb. We redirect the output to a file `minab.out`.

For the command line

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

the output file `minab.out` would be:

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

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

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

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

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

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

We can now repeat the interaction energy computation on the refined complex, using the same settings as for the raw PDB file in section 41.8.4:

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

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

```
======================================================================
Summary Statistics for Project MINPDB
Frames                : 1 to 1 (every 1)
```

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

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

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

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

The trajectory will go to the file `com.binpos`, specified as the third command line argument (the extension "`.binpos`" is appended automatically). We tether the Cα atoms with the same force as for the minimization in 41.8.5 through the last two arguments *`'::CA'`* and *`1.0`*. The GB option '1' is used (fourth argument for the `mdnab` command).

The file `md.out` will start with:

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

    mm_options:  cut=12.000000
    mm_options:  nsnb=25
    mm_options:  diel=C
    mm_options:  gb=1
    mm_options:  rgbmax=12.000000
    mm_options:  rattle=1
    mm_options:  dt=0.002000
    mm_options:  ntpr=101
    mm_options:  ntpr_md=10
    mm_options:  ntwx=0
    mm_options:  zerov=0
    mm_options:  tempi=50.000000
    mm_options:  temp0=100.000000
    mm_options:  gamma_ln=20.000000
    mm_options:  wcons=1.000000
constrained 214 atoms using expression ::CA
```

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

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

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

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

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

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

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

The command line is:

```
pymdpbsa --proj P38 --traj com.binpos --start 10 --stop 100       --step10&
```

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

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

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

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

Bibliography

[1] Pearlman, D.A.; Case, D.A.; Caldwell, J.W.; Ross, W.S.; Cheatham, T.E. III; DeBolt, S.; Ferguson, D.; Seibel, G.; Kollman, P. 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.*, **1995**, *91*, 1–41.

[2] Case, D.A.; Cheatham, T.; Darden, T.; Gohlke, H.; Luo, R.; Merz, K.M. Jr.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. The Amber biomolecular simulation programs. *J. Computat. Chem.*, **2005**, *26*, 1668–1688.

[3] Ponder, J.W.; Case, D.A. Force fields for protein simulations. *Adv. Prot. Chem.*, **2003**, *66*, 27–85.

[4] Cheatham, T.E.; Case, D.A. Twenty-five years of nucleic acid simulations. *Biopolymers*, **2013**, *99*, 969–977.

[5] Harvey, S.; McCammon, J.A. *Dynamics of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, 1987.

[6] Leach, A.R. *Molecular Modelling. Principles and Applications, Second Edition*. Prentice-Hall, Harlow, England, 2001.

[7] Cramer, C.J. *Essentials of Computational Chemistry: Theories and Models*. John Wiley & Sons, New York, 2002.

[8] Allen, M.P.; Tildesley, D.J. *Computer Simulation of Liquids*. Clarendon Press, Oxford, 1987.

[9] Frenkel, D.; Smit, B. *Understanding Molecular Simulation: From Algorithms to Applications. Second edition*. Academic Press, San Diego, 2002.

[10] Tuckerman, M.E. *Statistical Mechanics: Theory and Molecular Simulation*. Oxford University Press, Oxford, 2010.

[11] van Gunsteren, W.F.; Weiner, P.K.; Wilkinson, A.J. eds. *Computer Simulations of Biomolecular Systems, Vol. 3*. ESCOM Science Publishers, Leiden, 1997.

[12] Pratt, L.R.; Hummer, G. eds. *Simulation and Theory of Electrostatic Interactions in Solution*. American Institute of Physics, Melville, NY, 1999.

[13] Becker, O.; MacKerell, A.D.; Roux, B.; Watanabe, M. eds. *Computational Biochemistry and Biophysics*. Marcel Dekker, New York, 2001.

[14] Chipot, C.; Pohorille, A. eds. *Free energy calculations. Theory and Applications in Chemistry and Biology*. Springer, Berlin, 2007.

[15] Griebel, M.; Knapek, S.; Zumbusch, G. *Numerical Simulation in Molecular Dynamics. Numerical Algorithms, Parallelization, Applications*. Springer-Verlag, Berlin, 2010.

[16] Maier, J.A.; Martinez, C.; Kasavajhala, K.; Wickstrom, L.; Hauser, K.E.; Simmerling, C. ff14SB: Improving the Accuracy of Protein Side Chain and Backbone Parameters from ff99SB. *J. Chem. Theory Comput.*, **2015**, *11*, 3696–3713.

[17] Hornak, V.; Abel, R.; Okur, A.; Strockbine, B.; Roitberg, A.; Simmerling, C. Comparison of multiple Amber force fields and development of improved protein backbone parameters. *Proteins*, **2006**, *65*, 712–725.

[18] Graf, J.; Nguyen, P.H.; Stock, G.; Schwalbe, H. Structure and Dynamics of the Homologous Series of Alanine Peptides: A Joint Molecular Dynamics/NMR Study. *J. Am. Chem. Soc.*, **2007**, *129*, 1179–1189.

[19] Wickstrom, L.; Okur, A.; Simmerling, C. Evaluating the performance of the ff99SB force field based on NMR scalar coupling data. *Biophys. J.*, **2009**, *97*, 853–856.

[20] Nguyen, Hai; Roe, Daniel R.; Simmerling, Carlos. Improved Generalized Born Solvent Model Parameters for Protein Simulations. *J. Chem. Theory Comput.*, **2013**, *9*, 2020–2034.

[21] Debiec, K.T.; Cerutti, D.S.; Baker, L.R.; Gronenborn, A.M.; Case, D.A.; Chong, L.T. Further along the Road Less Traveled: AMBER ff15ipq, an Original Protein Force Field Built on a Self-Consistent Physical Model. *J. Chem. Theory Comput.*, **2016**, *12*, 3926–3947.

[22] Cerutti, D.S.; Rice, J.E.; Swope, W.C.; Case, D.A. Derivation of fixed partial charges for amino acids accommodating a specific water model and implicit polarization. *J. Phys. Chem. B*, **2103**, *117*, 2328–2338.

[23] Cerutti, D.S.; Swope, W.C.; Rice, J.E.; Case, D.A. ff14ipq: A Self-Consistent Force Field for Condensed-Phase Simulations of Proteins. *J. Chem. Theory Comput.*, **2014**, *10*, 4515–4534.

[24] Takemura, K.; Kitao, A. Water Model Tuning for Improved Reproduction of Rotational Diffusion and NMR Spectral Density. *J. Phys. Chem. B*, **2012**, *116*, 6279–6287.

[25] Cornell, W.D.; Cieplak, P.; Bayly, C.I.; Gould, I.R.; Merz, K.M. Jr.; Ferguson, D.M.; Spellmeyer, D.C.; Fox, T.; Caldwell, J.W.; Kollman, P.A. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.*, **1995**, *117*, 5179–5197.

[26] Wang, L.-P.; Martinez, T.J.; Pande, V.S. Building force fields: An automatic, systematic and reproducible approach. *J. Phys. Chem. Lett.*, **2014**, *5*, 1885–1891.

[27] Wang, L.; McKiernan, K.A.; Gomes, J.; Beauchamp, K.A.; Head-Gordon, T.; Rice, J.E.; Swope, W.C.; MartÃnez, T.J.; Pande, V.S. Building a More Predictive Protein Force Field: A Systematic and Reproducible Route to AMBER-FB15. *J. Phys. Chem. B*, **2017**, *121*, 4023–4039.

[28] Duan, Y.; Wu, C.; Chowdhury, S.; Lee, M.C.; Xiong, G.; Zhang, W.; Yang, R.; Cieplak, P.; Luo, R.; Lee, T. A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations. *J. Comput. Chem.*, **2003**, *24*, 1999–2012.

[29] Lee, M.C.; Duan, Y. Distinguish protein decoys by using a scoring function based on a new Amber force field, short molecular dynamics simulations, and the generalized Born solvent model. *Proteins*, **2004**, *55*, 620–634.

[30] Yang, L.; Tan, C.; Hsieh, M.-J.; Wang, J.; Duan, Y.; Cieplak, P.; Caldwell, J.; Kollman, P.A.; Luo, R. New-generation Amber united-atom force field. *J. Phys. Chem. B*, **2006**, *110*, 13166–13176.

[31] Cheatham, T.E. III; Young, M.A. Molecular dynamics simulation of nucleic acids: Successes, limitations and promise. *Biopolymers*, **2001**, *56*, 232–256.

[32] Cheatham, T.E. III. Simulation and modeling of nucleic acid structure, dynamics and interactions. *Curr. Opin. Struct. Biol.*, **2004**, *14*, 360–367.

[33] Varnai, P.; Djuranovic, D.; Lavery, R.; Hartmann, B. alpha/gamma Transitions in the B-DNA backbone. *Nucl. Acids Res.*, **2002**, *30*, 5398–5406.

[34] Spackova, N.; Cheatham, T.E.; Ryjacek, F.; Lankas, F.; vanMeervelt, L.; Hobza, P.; Sponer, J. Molecular Dynamics Simulations and Thermodynamics Analysis of DNA-Drug Complexes. Minor Groove Binding between 4',6-Diamidino-2-phenylindole and DNA Duplexes in Solution. *J. Am. Chem. Soc.*, **2003**, *125*, 1759–1769.

[35] Perez, A.; Marchan, I.; Svozil, D.; Sponer, J.; Cheatham, T.E.; Laughton, C.A.; Orozco, M. Refinement of the AMBER Force Field for Nucleic Acids: Improving the Description of alpha/gamma Conformers. *Biophys. J.*, **2007**, *92*, 3817–3829.

[36] Svozil, D.; Sponer, J.E.; Marchan, I.; Perez, A.; Cheatham, T.E.; Forti, F.; Luque, F.J.; Orozco, M.; Sponer, J. Geometrical and electronic structure variability of the sugar-phosphate backbone in nucleic acids. *J. Phys. Chem. B*, **2008**, *112*, 8188–8197.

[37] Banáš, P.; Hollas, D.; Zgarbová, M.; Jurecka, P.; Orozco, M.; Cheatham, T.E. III; Šponer, J.; Otyepka, M. Performance of molecular mechanics force fields for RNA simulations: Stability of UUCG and GNRA hairpins. *J. Chem. Theory. Comput.*, **2010**, *6*, 3836–3849.

[38] Zgarbova, M.; Otyepka, M.; Sponer, J.; Mladek, A.; Banas, P.; Cheatham, T.E.; Jurecka, P. Refinement of the Cornell et al. Nucleic Acids Force Field Based on Reference Quantum Chemical Calculations of Glycosidic Torsion Profiles. *J. Chem. Theory Comput.*, **2011**, *7*, 2886–2902.

[39] Yildirim, I.; Stern, H.A.; Kennedy, S.D.; Tubbs, J.D.; Turner, D.H. Reparameterization of RNA chi Torsion Parameters for the AMBER Force Field and Comparison to NMR Spectra for Cytidine and Uridine . *J. Chem. Theory Comput.*, **2010**, *6*, 1520–1531.

[40] Yildirim, I.; Stern, H.A.; Tubbs, J.D.; Kennedy, S.D.; Turner, D.H. Benchmarking AMBER Force Fields for RNA: Comparisons to NMR Spectra for Single-Stranded r(GACC) Are Improved by Revised chi Torsions. *J. Phys. Chem. B*, **2011**, *115*, 9261–9270.

[41] Yildirim, I.; Kennedy, S.D.; Stern, H.A.; Hart, J.M.; Kierzek, R.; Turner, D.H. Revision of AMBER Torsional Parameters for RNA Improves Free Energy Predictions for Tetramer Duplexes with GC and iGiC Base Pairs. *J. Chem. Theory Comput.*, **2012**, *8*, 172–181.

[42] Krepl, M.; Zgarbova, M.; Stadlbauer, P.; Otyepka, M.; Banas, P.; Koca, J.; Cheatham, T.E. III; Sponer, J. Reference simulations of noncanonical nucleic acids with different chi variants of the AMBER force field: Quadruplex DNA, quadruplex RNA, and Z-DNA. *J. Chem. Theory Comp.*, **2012**, *8*, 2506–2520.

[43] Aytenfisu, A.H.; Spasic, A.; Grossfield, A.; Stern, H.A.; Mathews, D.H. Revised RNA Dihedral Parameters for the Amber Force Field Improve RNA Molecular Dynamics. *J. Chem. Theory Comput.*, **2017**, *13*, 900–915.

[44] Aduri, R.; Psciuk, B.T.; Saro, P.; Taniga, H.; Schlegel, H.B.; SantaLucia, J. Jr. AMBER force field parameters for the naturally occurring modified nucleosides in RNA. *J. Chem. Theory Comput.*, **2007**, *3*, 1465–1475.

[45] Zgarbová, Marie; Luque, F. Javier; Šponer, Jiři; III, Thomas E. Cheatham; Otyepka, Michal; Jurečka, Petr. Toward improved description of dna backbone: Revisiting epsilon and zeta torsion force field parameters. *J. Chem. Theory Comput.*, **2013**, *9*, 2339–2354.

[46] Zgarbová, M.; Sponer, J.; Otyepka, M.; Cheatham, T.E. III; Galindo-Murillo, R.; Jurečka, P. Refinement of the Sugar-Phosphate Backbone Torsion Beta for AMBER Force Fields Improves the Description of Z- and B-DNA. *J. Chem. Theor. and Comp.*, **2015**, *12*, 5723–5736.

[47] Ivani, I.; Dans, P.D.; Noy, A.; Pérez, A.; Faustino, I.; Hopsital, A.; Walther, J.; Andrió, P.; Goni, R.; Balaceanu, A.; Portella, G.; Battistini, F.; GelpÃ, J.L.; González, C.; Vendruscolo, M.; Laughton, C.A.; Harris, S.; Case, D.A.; Orozco, M. Parmbsc1: A refined force field for DNA simulations. *Nature Meth.*, **2016**, *13*, 55–58.

[48] Galindo-Murillo, R.; Robertson, J.C.; Zgarbovic, M.; Sponer, J.; Otyepka, M.; Jureska, P.; Cheatham, T.E. Assessing the Current State of Amber Force Field Modifications for DNA. *J. Chem. Theory Comput.*, **2016**, *12*, 4114–4127.

[49] Kirschner, K.N.; Yongye, A.B.; Tschampel, S.M.; González-Outeiriño, J.; Daniels, C.R.; Foley, B.L.; Woods, R.J. GLYCAM06: A generalizable biomolecular force field. Carbohydrates. *J. Comput. Chem.*, **2008**, *29*, 622–655.

[50] DeMarco, M.L.; Woods, R.J. Atomic-resolution conformational analysis of the G(M3) ganglioside in a lipid bilayer and its implications for ganglioside-protein recognition at membrane surfaces. *Glycobiology*, **2009**, *19*, 344–355.

[51] DeMarco, M.L.; Woods, R.J.; Prestegard, J.H.; Tian, F. Presentation of Membrane-Anchored Glycosphingolipids Determined from Molecular Dynamics Simulations and NMR Paramagnetic Relaxation Rate Enhancement. *J. Am. Chem. Soc.*, **2010**, *132*, 1334–1338.

[52] Kadirvelraj, R.; Grant, O.C.; Goldstein, I.J.; Winter, H.C.; Tateno, H.; Fadda, E.; Woods, R.J. Structure and binding analysis of Polyporus squamosus lectin in complex with the Neu5Acα2-6Galβ1-4GlcNAc human-type influenza receptor. *Glycobiology*, **2011**, *21*, 973–984.

[53] DeMarco, M.L.; Woods, R.J. From agonist to antagonist: Structure and dynamics of innate immune glycoprotein MD-2 upon recognition of variably acylated bacterial endotoxins. *Mol. Immunol.*, **2011**, *49*, 124–133.

[54] Foley, B.L.; Tessier, M.B.; Woods, R.J. Carbohydrate force fields. *WIREs Comput. Mol. Sci.*, **2012**, *2*, 652–697.

[55] Ficko-Blean, E.; Stuart, C.P.; Suits, M.D.; Cid, M.; Tessier, M.; Woods, R.J.; Boraston, A.B. Carbohydrate Recognition by an Architecturally Complex α-N-Acetylglucosaminidase from Clostridium perfringens. *PLoS ONE*, **2012**, *7*, e33524.

[56] Tessier, M.B.; DeMarco, M.L.; Yongye, A.B.; Woods, R.J. Extension of the GLYCAM06 biomolecular force field to lipids, lipid bilayers and glycolipids. *Mol. Simul.*, **2008**, *34*, 349–363.

[57] Kirschner, K.N.; Woods, R.J. Solvent interactions determine carbohydrate conformation. *Proc. Natl. Acad. Sci. USA*, **2001**, *98*, 10541–10545.

[58] Woods, R.J. Restrained electrostatic potential charges for condensed phase simulations of carbohydrates. *J. Mol. Struct (Theochem)*, **2000**, *527*, 149–156.

[59] Woods, R.J. Derivation of net atomic charges from molecular electrstatic potentials. *J. Comput. Chem.*, **1990**, *11*, 29–310.

[60] Basma, M.; Sundara, S.; Calgan, D.; Venali, T.; Woods, R.J. Solvated ensemble averaging in the calculation of partial atomic charges. *J. Comput. Chem.*, **2001**, *22*, 1125–1137.

[61] Tschampel, S.M.; Kennerty, M.R.; Woods, R.J. TIP5P-consistent treatment of electrostatics for biomolecular simulations. *J. Chem. Theory Comput.*, **2007**, *3*, 1721–1733.

[62] DeMarco, M. L.; Woods, R. J. Bridging computational biology and glycobiology: A game of snakes and ladders. *Glycobiology*, **2008**, *18*, 426–440.

[63] Feller, S.E. Molecular dynamics simulations of lipid bilayers. *Curr. Opin. Colloid Interface Sci.*, **2000**, *5*, 217–223.

[64] Saiz, L.; Klein, M.L. Computer Simulation Studies of Model Biological Membranes. *Acc. Chem. Res.*, **2002**, *35*, 482–489.

[65] Lomize, A.; Pogozheva, I.; Lomize, M.; Mosberg, H. The role of hydrophobic interactions in positioning of peripheral proteins in membranes. *BMC Struct. Biol.*, **2007**, *7*, 44.

[66] Lundstrom, M. L K. H.; Chiu. *G Protein-Coupled Receptors in Drug Discovery*. Taylor & Francis, London, 2005.

[67] Crowley, M.F.; Williamson, M.J.; Walker, R.C. CHAMBER: Comprehensive support for CHARMM force fields within the AMBER software. *Int. J. Quant. Chem.*, **2009**, *109*, 3767–3772.

[68] Skjevik, Å.A.; Madej, Benjamin D.; Walker, R.C.; Teigen, K. Lipid11: A modular framework for lipid simulations using amber. *J. Phys. Chem. B*, **2012**, *116*, 11124–11136.

[69] Dickson, C.J.; Rosso, L.; Betz, R.M.; Walker, R.C.; Gould, I.R. GAFFlipid: a General Amber Force Field for the accurate molecular dynamics simulation of phospholipid. *Soft Matter*, **2012**, *8*, 9617.

[70] Dickson, C.J.; Madej, B.D.; Skjevik, A.A.; Betz, R.M.; Teigen, K.; Gould, I.R.; Walker, R.C. Lipid14: The Amber Lipid Force Field. *J. Chem. Theory Comput.*, **2014**, *10*, 865–879.

[71] Skjevik, Å.A.; Madej, B.D.; Dickson, C.J.; Teigen, K.; Walker, R.C.; Gould, I.R. All-atom lipid bilayer self-assembly with the amber and charmm lipid force fields. *Chem. Commun.*, **2015**, *51*, 4402–4405.

[72] Skjevik, Å.A.; Madej, B.D.; Dickson, C.J.; Lin, C.; Teigen, K.; Walker, R.C.; Gould, I.R. Simulations of lipid bilayer self-assembly using all-atom lipid force fields. *Phys. Chem. Chem. Phys.*, **2016**, *18*, 10573–10584.

[73] Madej, B.D.; Gould, I.R.; Walker, R.C. A Parameterization of Cholesterol for Mixed Lipid Bilayer Simulation within the Amber Lipid14 Force Field. *J Phys Chem B*, **2015**, *119*, 12424–12435.

[74] Jorgensen, W.L.; Chandrasekhar, J.; Madura, J.; Klein, M.L. Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.*, **1983**, *79*, 926–935.

[75] Price, D.J.; Brooks, C.L. A modified TIP3P water potential for simulation with Ewald summation. *J. Chem. Phys.*, **2004**, *121*, 10096–10103.

[76] Jorgensen, W.L.; Madura, J.D. Temperature and size dependence for Monte Carlo simulations of TIP4P water. *Mol. Phys.*, **1985**, *56*, 1381–1392.

[77] Horn, H.W.; Swope, W.C.; Pitera, J.W.; Madura, J.D.; Dick, T.J.; Hura, G.L.; Head-Gordon, T. Development of an improved four-site water model for biomolecular simulations: TIP4P-Ew. *J. Chem. Phys.*, **2004**, *120*, 9665–9678.

[78] Horn, H.W.; Swope, W.C.; Pitera, J.W. Characterization of the TIP4P-Ew water model: Vapor pressure and boiling point. *J. Chem. Phys.*, **2005**, *123*, 194504.

[79] Mahoney, M.W.; Jorgensen, W.L. A five-site model for liquid water and the reproduction of the density anomaly by rigid, nonpolarizable potential functions. *J. Chem. Phys.*, **2000**, *112*, 8910–8922.

[80] Izadi, S.; Anandakrishnan, R.; Onufriev, A.V. Building Water Models: A Different Approach. *J. Phys. Chem. Lett.*, **2014**, *5*, 3863–3871.

[81] Izadi, Saeed; Onufriev, Alexey V. Accuracy limit of rigid 3-point water models. *J. Chem. Phys.*, **2016**, *145*, 074501.

[82] Caldwell, J.W.; Kollman, P.A. Structure and properties of neat liquids using nonadditive molecular dynamics: Water, methanol and N-methylacetamide. *J. Phys. Chem.*, **1995**, *99*, 6208–6219.

[83] Berendsen, H.J.C.; Grigera, J.R.; Straatsma, T.P. The missing term in effective pair potentials. *J. Phys. Chem.*, **1987**, *91*, 6269–6271.

[84] Wu, Y.; Tepper, H.L.; Voth, G.A. Flexible simple point-charge water model with improved liquid-state properties. *J. Chem. Phys.*, **2006**, *124*, 024503.

[85] Paesani, F.; Zhang, W.; Case, D.A.; Cheatham, T.E.; Voth, G.A. An accurate and simple quantum model for liquid water. *J. Chem. Phys.*, **2006**, *125*, 184507.

[86] Cieplak, P.; Caldwell, J.; Kollman, P. Molecular mechanical models for organic and biological systems going beyond the atom centered two body additive approximation: Aqueous solution free energies of methanol and N-methyl acetamide, nucleic acid base, and amide hydrogen bonding and chloroform/water partition coefficients of the nucleic acid bases. *J. Comput. Chem.*, **2001**, *22*, 1048–1057.

[87] Anandakrishnan, R.; Baker, C.; Izadi, S.; Onufriev, A. V. Point charges optimally placed to represent the multipole expansion of charge distributions. *PloS one*, **2013**, *8*, e67715.

[88] Niu, Shuqiang; Tan, Ming L.; Ichiye, Toshiko. The large quadrupole of water molecules. *J. Chem. Phys.*, **2011**, *134*, 134501+.

[89] Mukhopadhyay, A.; Fenley, A. T.; Tolokh, I. S.; Onufriev, A. V. Charge hydration asymmetry: the basic principle and how to use it to test and improve water models. *J. Phys. Chem. B*, **2012**, *116*, 9776–9783.

[90] Bergonzo, Christina; III, Thomas E. Cheatham. Improved force field parameters lead to a better description of rna structure. *J. of Chem. Theory Comput.*, **2015**, *11*, 3969–3972.

[91] Gao, Kaifu; Yin, Jian; Henriksen, Niel M.; Fenley, Andrew T.; Gilson, Michael K. Binding enthalpy calculations for a neutral host-guest pair yield widely divergent salt effects across water models. *J. of Chem. Theory Comput.*, **2015**, *11*, 4555–4564.

[92] Joung, I.S.; Cheatham, T.E. III. Molecular dynamics simulations of the dynamic and energetic properties of alkali and halide ions using water-model-specific ion parameters. *J. Phys. Chem. B*, **2009**, *113*, 13279–13290.

[93] Joung, S.; Cheatham, T.E. III. Determination of alkali and halide monovalent ion parameters for use in explicitly solvated biomolecular simulations. *J. Phys. Chem. B*, **2008**, *112*, 9020–9041.

[94] Li, P.; Roberts, B.P.; Chakravorty, D.K.; Merz, K.M. Jr. Rational Design of Particle Mesh Ewald Compatible Lennard-Jones Parameters for +2 Metal Cations in Explicit Solvent. *J. Chem. Theory Comput.*, **2013**, *9*, 2733–2748.

[95] Li, P.; Merz, K.M. Jr. Taking into Account the Ion-Induced Dipole Interaction in the Nonbonded Model of Ions. *J. Chem. Theory Comput.*, **2014**, *10*, 289–297.

[96] Li, P.; Song, L.F.; Merz, K.M. Jr. Parameterization of Highly Charged Metal Ions Using the 12-6-4 LJ-Type Nonbonded Model in Explicit Water. *J. Phys. Chem. B*, **2015**, *119*, 883–895.

[97] Li, P.; Song, L.F.; Merz, K.M. Jr. Systematic Parameterization of Monovalent Ions Employing the Nonbonded Model. *J. Chem. Theory Comput.*, **2015**, *11*, 1645–1657.

[98] Panteva, M. T.; Giambasu, G. M.; York, D. M. Comparison of Structural, Thermodynamic, Kinetic and Mass Transport Properties of Mg2+ Models Commonly Used in Biomolecular Simulations. *J. Comput. Chem.*, **2015**, *36*, 970–982.

[99] Panteva, M. T.; Giambasu, G. M.; York, D. M. Force Field for Mg2+, Mn2+, Zn2+ and Cd2+ Ions That Have Balanced Interactions with Nucleic Acids. *J. Phys. Chem. B*, **2015**, *119*, 15460–15470.

[100] Homeyer, N.; Horn, A.H.C.; Lanig, H.; Sticht, H. AMBER force-field parameters for phosphorylated amino acids in different protonation states: phosphoserine, phosphothreonine, phosphotyrosine, and phosphohistidine. *J. Mol. Model.*, **2006**, *12*, 281–289.

[101] Steinbrecher, T.; Latzer, J.; Case, D.A. Revised AMBER Parameters for Bioorganic Phosphates. *J. Chem. Theory Comput.*, **2012**, *8*, 4405–4412.

[102] Blood, Dalton L.; Rosnik, Andreana M.; Krueger, Brent P. Molecular dynamics parameters for the gfp chromophore and some of its analogues. *Manuscript in preparation*, **2016**.

[103] Wollacott, A.M.; Merz, K.M. Jr. Development of a parameterized force field to reproduce semiempirical geometries. *J. Chem. Theory Comput.*, **2006**, *2*, 1070–1077.

[104] Weiner, S.J.; Kollman, P.A.; Case, D.A.; Singh, U.C.; Ghio, C.; Alagona, G.; Profeta, S. Jr.; Weiner, P. A new force field for molecular mechanical simulation of nucleic acids and proteins. *J. Am. Chem. Soc.*, **1984**, *106*, 765–784.

[105] Weiner, S.J.; Kollman, P.A.; Nguyen, D.T.; Case, D.A. An all-atom force field for simulations of proteins and nucleic acids. *J. Comput. Chem.*, **1986**, *7*, 230–252.

[106] Singh, U.C.; Weiner, S.J.; Kollman, P.A. Molecular dynamics simulations of d(C-G-C-G-A).d(T-C-G-C-G) with and without "hydrated" counterions. *Proc. Nat. Acad. Sci.*, **1985**, *82*, 755–759.

[107] Kollman, P.A.; Dixon, R.; Cornell, W.; Fox, T.; Chipot, C.; Pohorille, A. in *Computer Simulation of Biomolecular Systems, Vol. 3*, Wilkinson, A.; Weiner, P.; van Gunsteren, W.F., Eds., pp 83–96. Elsevier, 1997.

[108] Beachy, M.D.; Friesner, R.A. Accurate ab intio quantum chemical determination of the relative energies of peptide conformations and assessment of empirical force fields. *J. Am. Chem. Soc.*, **1997**, *119*, 5908–5920.

[109] Wang, L.; Duan, Y.; Shortle, R.; Imperiali, B.; Kollman, P.A. Study of the stability and unfolding mechanism of BBA1 by molecular dynamics simulations at different temperatures. *Prot. Sci.*, **1999**, *8*, 1292–1304.

[110] Higo, J.; Ito, N.; Kuroda, M.; Ono, S.; Nakajima, N.; Nakamura, H. Energy landscape of a peptide consisting of α-helix, 3_{10} helix, β-turn, β-hairpin and other disordered conformations. *Prot. Sci.*, **2001**, *10*, 1160–1171.

[111] Cheatham, T.E. III; Cieplak, P.; Kollman, P.A. A modified version of the Cornell et al. force field with improved sugar pucker phases and helical repeat. *J. Biomol. Struct. Dyn.*, **1999**, *16*, 845–862.

[112] Wang, J.; Cieplak, P.; Kollman, P.A. How well does a restrained electrostatic potential (RESP) model perform in calculating conformational energies of organic and biological molecules? *J. Comput. Chem.*, **2000**, *21*, 1049–1074.

[113] Cieplak, P.; Cornell, W.D.; Bayly, C.; Kollman, P.A. Application of the multimolecule and multiconformational RESP methodology to biopolymers: Charge derivation for DNA, RNA and proteins. *J. Comput. Chem.*, **1995**, *16*, 1357–1377.

[114] Cieplak, P.; Dupradeau, F.-Y.; Duan, Y.; Wang, J. Polarization effects in molecular mechanical force fields. *J. Phys.: Condens. Matter*, **2009**, *21*, 333102.

[115] Wang, Z.-X.; Zhang, W.; Wu, C.; Lei, H.; Cieplak, P.; Duan, Y. Strike a Balance: Optimization of backbone torsion parameters of AMBER polarizable force field for simulations of proteins and peptides. *J. Comput. Chem.*, **2006**, *27*, 781–790.

[116] Dixon, R.W.; Kollman, P.A. Advancing beyond the atom-centered model in additive and nonadditive molecular mechanics. *J. Comput. Chem.*, **1997**, *18*, 1632–1646.

[117] Meng, E.; Cieplak, P.; Caldwell, J.W.; Kollman, P.A. Accurate solvation free energies of acetate and methylammonium ions calculated with a polarizable water model. *J. Am. Chem. Soc.*, **1994**, *116*, 12061–12062.

[118] Åqvist, J. Ion-water interaction potentials derived from free energy perturbation simulations. *J. Phys. Chem.*, **1990**, *94*, 8021–8024.

[119] Dang, L. Mechanism and thermodynamics of ion selectivity in aqueous solutions of 18-crown-6 ether: A molecular dynamics study. *J. Am. Chem. Soc.*, **1995**, *117*, 6954–6960.

[120] Auffinger, P.; Cheatham, T.E. III; Vaiana, A.C. Spontaneous formation of KCl aggregates in biomolecular simulations: a force field issue? *J. Chem. Theory Comput.*, **2007**, *3*, 1851–1859.

[121] David, Laurent; Luo, Ray; Gilson, Michael K. Comparison of generalized born and poisson models: Energetics and dynamics of hiv protease. *J. Comput. Chem.*, **2000**, *21*, 295–309.

[122] Feig, Michael. Kinetics from Implicit Solvent Simulations of Biomolecules as a Function of Viscosity. *J. Chem. Theory Comput.*, **2007**, *3*, 1734–1748.

[123] Amaro, Rommie E.; Cheng, Xiaolin; Ivanov, Ivaylo; Xu, Dong; Mccammon, Andrew J. Characterizing Loop Dynamics and Ligand Recognition in Human- and Avian-Type Influenza Neuraminidases via Generalized Born Molecular Dynamics and End-Point Free Energy Calculations. *J. Am. Chem. Soc.*, **2009**, *131*, 4702–4709.

[124] Zagrovic, Bojan; Pande, Vijay. Solvent viscosity dependence of the folding rate of a small protein: Distributed computing study. *J. Comput. Chem.*, **2003**, *24*, 1432–1436.

[125] Anandakrishnan, Ramu; Drozdetski, Aleksander; Walker, Ross C.; Onufriev, Alexey V. Speed of Conformational Change: Comparing Explicit and Implicit Solvent Molecular Dynamics Simulations. *Biophysical Journal*, **2015**, *108*, 1153–1164.

[126] Weiser, J.; Shenkin, P.S.; Still, W.C. Approximate Atomic Surfaces from Linear Combinations of Pairwise Overlaps (LCPO). *J. Comput. Chem.*, **1999**, *20*, 217–230.

[127] Still, W.C.; Tempczyk, A.; Hawley, R.C.; Hendrickson, T. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.*, **1990**, *112*, 6127–6129.

[128] Schaefer, M.; Karplus, M. A comprehensive analytical treatment of continuum electrostatics. *J. Phys. Chem.*, **1996**, *100*, 1578–1599.

[129] Edinger, S.R.; Cortis, C.; Shenkin, P.S.; Friesner, R.A. Solvation free energies of peptides: Comparison of approximate continuum solvation models with accurate solution of the Poisson-Boltzmann equation. *J. Phys. Chem. B*, **1997**, *101*, 1190–1197.

[130] Jayaram, B.; Sprous, D.; Beveridge, D.L. Solvation free energy of biomacromolecules: Parameters for a modified generalized Born model consistent with the AMBER force field. *J. Phys. Chem. B*, **1998**, *102*, 9571–9576.

[131] Cramer, C.J.; Truhlar, D.G. Implicit solvation models: Equilibria, structure, spectra, and dynamics. *Chem. Rev.*, **1999**, *99*, 2161–2200.

[132] Bashford, D.; Case, D.A. Generalized Born models of macromolecular solvation effects. *Annu. Rev. Phys. Chem.*, **2000**, *51*, 129–152.

[133] Onufriev, A.; Bashford, D.; Case, D.A. Modification of the generalized Born model suitable for macromolecules. *J. Phys. Chem. B*, **2000**, *104*, 3712–3720.

[134] Lee, M.S.; Salsbury, F.R. Jr.; Brooks, C.L. III. Novel generalized Born methods. *J. Chem. Phys.*, **2002**, *116*, 10606–10614.

[135] Dominy, B.N.; Brooks, C.L. III. Development of a generalized Born model parameterization for proteins and nucleic acids. *J. Phys. Chem. B*, **1999**, *103*, 3765–3773.

[136] Tsui, V.; Case, D.A. Molecular dynamics simulations of nucleic acids using a generalized Born solvation model. *J. Am. Chem. Soc.*, **2000**, *122*, 2489–2498.

[137] Calimet, N.; Schaefer, M.; Simonson, T. Protein molecular dynamics with the generalized Born/ACE solvent model. *Proteins*, **2001**, *45*, 144–158.

[138] Onufriev, A.; Bashford, D.; Case, D.A. Exploring protein native states and large-scale conformational changes with a modified generalized Born model. *Proteins*, **2004**, *55*, 383–394.

[139] Srinivasan, J.; Trevathan, M.W.; Beroza, P.; Case, D.A. Application of a pairwise generalized Born model to proteins and nucleic acids: inclusion of salt effects. *Theor. Chem. Acc.*, **1999**, *101*, 426–434.

[140] Onufriev, A.; Case, D.A.; Bashford, D. Effective Born radii in the generalized Born approximation: The importance of being perfect. *J. Comput. Chem.*, **2002**, *23*, 1297–1304.

[141] Hawkins, G.D.; Cramer, C.J.; Truhlar, D.G. Parametrized models of aqueous free energies of solvation based on pairwise descreening of solute atomic charges from a dielectric medium. *J. Phys. Chem.*, **1996**, *100*, 19824–19839.

[142] Richards, F.M. Areas, volumes, packing, and protein structure. *Ann. Rev. Biophys. Bioeng.*, **1977**, *6*, 151–176.

[143] Schaefer, M.; Froemmel, C. A precise analytical method for calculating the electrostatic energy of macromolecules in aqueous solution. *J. Mol. Biol.*, **1990**, *216*, 1045–1066.

[144] Feig, M.; Onufriev, A.; Lee, M.; Im, W.; Case, D.A.; Brooks, C.L. III. Performance comparison of the generalized Born and Poisson methods in the calculation of the electrostatic solvation energies for protein structures. *J. Comput. Chem.*, **2004**, *25*, 265–284.

[145] Geney, R.; Layten, M.; Gomperts, R.; Simmerling, C. Investigation of salt bridge stability in a generalized Born solvent model. *J. Chem. Theory Comput.*, **2006**, *2*, 115–127.

[146] Okur, A.; Wickstrom, L.; Simmerling, C. Evaluation of salt bridge structure and energetics in peptides using explicit, implicit and hybrid solvation models. *J. Chem. Theory Comput.*, **2008**, *4*, 488–498.

[147] Okur, A.; Wickstrom, L.; Layten, M.; Geney, R.; Song, K.; Hornak, V.; Simmerling, C. Improved efficiency of replica exchange simulations through use of a hybrid explicit/implicit solvation model. *J. Chem. Theory Comput.*, **2006**, *2*, 420–433.

[148] Roe, D.R.; Okur, A.; Wickstrom, L.; Hornak, V.; Simmerling, C. Secondary Structure Bias in Generalized Born Solvent Models: Comparison of Conformational Ensembles and Free Energy of Solvent Polarization from Explicit and Implicit Solvation. *J. Phys. Chem. B*, **2007**, *111*, 1846–1857.

[149] Nguyen, Hai; Maier, James; Huang, He; Perrone, Victoria; Simmerling, Carlos. Folding simulations for proteins with diverse topologies are accessible in days with a physics-based force field and implicit solvent. *J. Am. Chem. Soc.*, **2014**, *136*, 13959–13962. PMID: 25255057.

[150] Nguyen, H.; Pérez, A.; Bermeo, S.; Simmerling, C. Refinement of Generalized Born Implicit Solvation Parameters for Nucleic Acids and Their Complexes with Proteins. *J. Chem. Theory Comput.*, **2015**, *11*, 3714–3728.

[151] Onufriev, A. in *Modeling Solvent Environments*, Feig, M, Ed., (Wiley, USA). pp 127–165. 2010.

[152] Hawkins, G.D.; Cramer, C.J.; Truhlar, D.G. Pairwise solute descreening of solute charges from a dielectric medium. *Chem. Phys. Lett.*, **1995**, *246*, 122–129.

[153] Schaefer, M.; Van Vlijmen, H.W.T.; Karplus, M. Electrostatic contributions to molecular free energies in solution. *Adv. Protein Chem.*, **1998**, *51*, 1–57.

[154] Tsui, V.; Case, D.A. Theory and applications of the generalized Born solvation model in macromolecular simulations. *Biopolymers (Nucl. Acid. Sci.)*, **2001**, *56*, 275–291.

[155] Sosa, C.P.; Hewitt, T.; Lee, M.S.; Case, D.A. Vectorization of the generalized Born model for molecular dynamics on shared-memory computers. *J. Mol. Struct. (Theochem)*, **2001**, *549*, 193–201.

[156] Mongan, J.; Simmerling, C.; A. McCammon, J.; A. Case, D.; Onufriev, A. Generalized Born with a simple, robust molecular volume correction. *J. Chem. Theory Comput.*, **2007**, *3*, 156–169.

[157] Sitkoff, D.; Sharp, K.A.; Honig, B. Accurate calculation of hydration free energies using macroscopic solvent models. *J. Phys. Chem.*, **1994**, *98*, 1978–1988.

[158] Sigalov, G.; Scheffel, P.; Onufriev, A. Incorporating variable environments into the generalized Born model. *J. Chem. Phys.*, **2005**, *122*, 094511.

[159] Sigalov, G.; Fenley, A.; Onufriev, A. Analytical electrostatics for biomolecules: Beyond the generalized Born approximation . *J. Chem. Phys.*, **2006**, *124*, 124902.

[160] Aguilar, B.; Onufriev, A. V. Efficient computation of the total solvation energy of small molecules via the r6 generalized born model. *J. Chem. Theory Comput.*, **2012**, *8*, 2404–2411.

[161] Aguilar, B.; Shadrach, R.; Onufriev, A. V. Reducing the secondary structure bias in the generalized born model via r6 effective radii. *J. Chem. Theory Comput.*, **2010**, *6*, 3613–3630.

[162] Forouzesh, Negin; Izadi, Saeed; Onufriev, Alexey V. Grid-based molecular surface GB model for single-point calculations of electrostatic solvation free energies. *Manuscript in preparation*, **2017**.

[163] Mukhopadhyay, A.; Aguilar, B. H.; Tolokh, I. S.; Onufriev, A. V. Introducing charge hydration asymmetry into the generalized born model. *J Chem. Theory Comput.*, **2014**, *10*, 1788–1794.

[164] Cai, Q.; Ye, X.; Wang, J.; Luo, R. On-the-Fly Numerical Surface Integration for Finite-Difference Poisson-Boltzmann Methods. *J. Chem. Theory Comput.*, **2011**, *7*, 3608–3619.

[165] Luo, R.; David, L.; Gilson, M.K. Accelerated Poisson-Boltzmann calculations for static and dynamic systems. *J. Comput. Chem.*, **2002**, *23*, 1244–1253.

[166] Wang, Jun; Luo, Ray. Assessment of Linear Finite-Difference Poisson-Boltzmann solvers. *J. Comput. Chem.*, **2010**, *31*, 1689–1698.

[167] Cai, Q.; Hsieh, M.-J.; Wang, J.; Luo, R. Performance of Nonlinear Finite-Difference Poisson-Boltzmann Solvers. *J. Chem. Theory Comput.*, **2010**, *6*, 203–211.

[168] Honig, B.; Nicholls, A. Classical electrostatics in biology and chemistry. *Science*, **1995**, *268*, 1144–1149.

[169] Lu, Q.; Luo, R. A Poisson-Boltzmann dynamics method with nonperiodic boundary condition. *J. Chem. Phys.*, **2003**, *119*, 11035–11047.

[170] Gilson, M.K.; Sharp, K.A.; Honig, B.H. Calculating the electrostatic potential of molecules in solution: method. *J. Comput. Chem.*, **1988**, *9*, 327–35.

[171] Warwicker, J.; Watson, H.C. Calculation of the electric potential in the active site cleft due to. *J. Mol. Biol.*, **1982**, *157*, 671–679.

[172] Klapper, I.; Hagstrom, R.; Fine, R.; Sharp, K.; Honig, B. Focussing of electric fields in the active stie of Cu, Zn superoxide dismutase. *Proteins*, **1986**, *1*, 47–59.

[173] Tan, C. H.; Tan, Y. H.; Luo, R. Implicit nonpolar solvent models. *J. Phys. Chem. B*, **2007**, *111*, 12263–12274.

[174] Gallicchio, E.; Kubo, M.M.; Levy, R.M. Enthalpy-entropy and cavity decomposition of alkane hydration free energies: Numerical results and implications for theories of hydrophobic solvation. *J. Phys. Chem.*, **2000**, *104*, 6271–6285.

[175] Floris, F.; Tomasi, J. Evaluation of the dispersion contribution to the solvation energy. A simple computational model in the continuum approximation. *J. Comput. Chem.*, **1989**, *10*, 616–627.

[176] Davis, M.E.; McCammon, J.A. Solving the finite-difference linearized Poisson-Boltzmann equation – a comparison of relaxation and conjugate gradient methods. *J. Comput. Chem.*, **1989**, *10*, 386–391.

[177] Nicholls, A.; Honig, B. A rapid finite difference algorithm, utilizing successive over-relaxation to solve the Poisson-Boltzmann equation. *J. Comput. Chem.*, **1991**, *12*, 435–445.

[178] Bashford, D. An object-oriented programming suite for electrostatic effects in biological molecules. *Lect. Notes Comput. Sci.*, **1997**, *1343*, 233–240.

[179] Wang, J.; Cai, Q.; Li, Z.; Zhao, H.; Luo, R. Achieving Energy Conservation in Poisson-Boltzmann Molecular Dynamics: Accuracy and Precision with Finite-difference Algorithms. *Chem. Phys. Lett.*, **2009**, *468*, 112.

[180] Li, Z.; K., Ito. *The Immbersed Interface Method: numberical sollutions of PDEs involving interfaces and irregular domains*. SIAM Frontiers in Applied Mathematics, Philadelphia, 2006.

[181] Luty, B.A.; Davis, M.E.; McCammon, J.A. Electrostatic energy calculations by a finite-difference method: Rapid calculation of charge-solvent interaction energies. *J. Comput. Chem.*, **1992**, *13*, 768–771.

[182] Cai, Q.; Wang, J.; Zhao, H.; Luo, R. On removal of charge singularity in Poisson-Boltzmann equation. *J. Chem. Phys.*, **2009**, *130*, 145101.

[183] Cai, Q.; Ye, X.; Wang, J.; Luo, R. Dielectric boundary force in numerical Poisson-Boltzmann methods: Theory and numerical strategies. *Chem. Phys. Lett.*, **2011**, *514*, 368–373.

[184] Davis, M.E.; McCammon, J.A. Dielectric boundary smoothing in finite difference solutions of the Poisson equation: An approach to improve accuracy and convergence. *J. Comput. Chem.*, **1991**, *12*, 909–912.

[185] Wang, J.; Cai, Q.; Xiang, Y.; Luo, R. Reducing Grid Dependence in Finite-Difference Poisson-Boltzmann Calculations. *J. Chem. Theory Comput.*, **2012**, *8*, 2741–2751.

[186] Davis, M.E.; McCammon, J.A. Electrostatics in biomolecular structure and dynamics. *Chem. Rev.*, **1990**, *90*, 509–521.

[187] Tan, C. H.; Yang, L. J.; Luo, R. How well does Poisson-Boltzmann implicit solvent agree with explicit solvent? A quantitative analysis. *J. Phys. Chem. B*, **2006**, *110*, 18680–18687.

[188] Ye, X.; Wang, J.; Luo, R. A Revised Density Function for Molecular Surface Calculation in Continuum Solvent Models. *J. Chem. Theory Comput.*, **2010**, *6*, 1157–1169.

[189] Cai, Q.; Ye, X.; Luo, R. Dielectric Pressure in Continuum Electrostatic Solvation of Biomolecules. *Phys. Chem. Chem. Phys.*, **2012**, *14*, 15917–15925.

[190] Hsieh, M.-J.; Luo, R. Exploring a coarse-grained distributive strategy for finite-difference poisson-boltzmann calculations. *J. Molec. Model.*, **2011**.

[191] Gilson, M.K.; Davis, M.E; Luty, B.A.; McCammon, J.A. Computation of electrostatic forces on solvated molecules using the Poisson-Boltzmann equation. *J Phys Chem*, **1993**, *97*, 3591–3600.

[192] Qi, R.; Botello-Smith, W.; Luo, R. Acceleration of Linear Finite-Difference Poisson-Boltzmann Methods on Graphics Processing Units. **2017**. arXiv:1704.02745.

[193] Luchko, T.; Gusarov, S.; Roe, D. R.; Simmerling, C.; Case, D. A.; Tuszynski, J.; Kovalenko, A. Three-dimensional molecular theory of solvation coupled with molecular dynamics in Amber. *J. Chem. Theory Comput.*, **2010**, *6*, 607–624.

[194] Chandler, D.; Andersen, H. C. Optimized cluster expansions for classical fluids. ii. theory of molecular liquids. *J. Chem. Phys.*, **1972**, *57*, 1930–1937.

[195] Hirata, F.; Rossky, P. J. An extended RISM equation for molecular polar fluids. *Chem. Phys. Lett.*, **1981**, pp 329–334.

[196] Hirata, F.; Pettitt, B. M.; Rossky, P. J. Application of an extended rism equation to dipolar and quadrupolar fluids. *J. Chem. Phys.*, **1982**, *77*, 509–520.

[197] Hirata, F.; Rossky, P. J.; Pettitt, B. M. The interionic potential of mean force in a molecular polar solvent from an extended rism equation. *J. Chem. Phys.*, **1983**, *78*, 4133–4144.

[198] Chandler, D.; McCoy, J.; Singer, S. Density functional theory of nonuniform polyatomic systems. i. general formulation. *J. Chem. Phys.*, **1986**, *85*, 5971–5976.

[199] Chandler, D.; McCoy, J.; Singer, S. Density functional theory of nonuniform polyatomic systems. ii. rational closures for integral equations. *J. Chem. Phys.*, **1986**, *85*, 5977–5982.

[200] Beglov, D.; Roux, B. Numerical solution of the hypernetted chain equation for a solute of arbitrary geometry in three dimensions. *J. Chem. Phys.*, **1995**, *103*, 360–364.

[201] Beglov, D.; Roux, B. An integral equation to describe the solvation of polar molecules in liquid water. *J. Phys. Chem. B*, **1997**, *101*, 7821–7826.

[202] Kovalenko, A.; Hirata, F. Three-dimensional density profiles of water in contact with a solute of arbitrary shape: a RISM approach. *Chem. Phys. Lett.*, **1998**, *290*, 237–244.

[203] Kovalenko, A.; Hirata, F. Self-consistent description of a metal–water interface by the Kohn–Sham density functional theory and the three-dimensional reference interaction site model. *J. Chem. Phys.*, **1999**, *110*, 10095–10112.

[204] Kovalenko, A. In Hirata [796], chapter 4.

[205] Kovalenko, A.; Hirata, F. Potentials of mean force of simple ions in ambient aqueous solution. i: Three-dimensional reference interaction site model approach. *J. Chem. Phys.*, **2000**, *112*, 10391–10402.

[206] Kovalenko, A.; Hirata, F. Potentials of mean force of simple ions in ambient aqueous solution. ii: Solvation structure from the three-dimensional reference interaction site model approach, and comparison with simulations. *J. Chem. Phys.*, **2000**, *112*, 10403–10417.

[207] Hansen, J.-P.; McDonald, I. R. *Theory of simple liquids*. Academic Press, London, 1990.

[208] Hirata, F. In *Molecular Theory of Solvation* [796], chapter 1.

[209] Perkyns, J. S.; Pettitt, B. M. A site-site theory for finite concentration saline solutions. *J. Chem. Phys.*, **1992**, *97*, 7656–7666.

[210] Kovalenko, A.; Ten-No, S.; Hirata, F. Solution of three-dimensional reference interaction site model and hypernetted chain equations for simple point charge water by modified method of direct inversion in iterative subspace. *J. Comput. Chem*, **1999**, *20*, 928–936.

[211] Joung, In Suk; Luchko, Tyler; Case, David A. Simple electrolyte solutions: Comparison of DRISM and molecular dynamics results for alkali halide solutions. *J Chem Phys*, **2013**, *138*, 044103.

[212] Giambasu, George M.; Luchko, Tyler; Herschlag, Daniel; York, Darrin M.; Case, David A. Ion counting from explicit-solvent simulations and 3d-RISM. *Biophys J*, **2014**, *106*, 883–894.

[213] Kaminski, J. W.; Gusarov, S.; Wesolowski, T. A.; Kovalenko, Andiry. Modeling solvatochromic shifts using the orbital-free embedding potential at statistically mechanically averaged solvent density. *J. Phys. Chem. A*, **2010**, *114*, 6082–6096.

[214] Frigo, M.; Johnson, S. G. FFTW: An adaptive software architecture for the FFT. in *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pp 1381–1384. IEEE, 1998.

[215] Frigo, M. A fast Fourier transform compiler. in *Proc. 1999 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, volume 34, pp 169–180. ACM, 1999.

[216] Singer, S. J.; Chandler, D. Free energy functions in the extended RISM approximation. *Mol. Phys.*, **1985**, *55*, 621–625.

[217] Pettitt, B. M.; Rossky, P. J. Alkali halides in water: Ion-solvent correlations and ion-ion potentials of mean force at infinite dilution. *J. Chem. Phys.*, **1986**, *15*, 5836–5844.

[218] Kast, Stefan M. Free energies from integral equation theories: Enforcing path independence. *Phys. Rev. E*, **2003**, *67*, 041203.

[219] Schmeer, G.; Maurer, A. Development of thermodynamic properties of electrolyte solutions with the help of RISM-calculations at the Born-Oppenheimer level. *Phys. Chem. Chem. Phys.*, **2010**, *12*, 2407–2417.

[220] Gusarov, S.; Ziegler, T.; Kovalenko, A. Self-consistent combination of the three-dimensional RISM theory of molecular solvation with analytical gradients and the amsterdam density functional package. *J. Phys. Chem. A*, **2006**, *110*, 6083–6090.

[221] Miyata, T.; Hirata, F. Combination of molecular dynamics method and 3D-RISM theory for conformational sampling of large flexible molecules in solution. *J. Comput. Chem.*, **2007**, *29*, 871–882.

[222] Kast, Stefan M.; Kloss, Thomas. Closed-form expressions of the chemical potential for integral equation closures with certain bridge functions. *J. Chem. Phys.*, **2008**, *129*, 236101.

[223] Chandler, David; Singh, Yashwant; Richardson, Diane M. Excess electrons in simple fluids. I. General equilibrium theory for classical hard sphere solvents. *J. Chem. Phys.*, **1984**, *81*, 1975–1982.

[224] Ichiye, Toshiko; Chandler, David. Hypernetted chain closure reference interaction site method theory of structure and thermodynamics for alkanes in water. *J. Phys. Chem.*, **1988**, *92*, 5257–5261.

[225] Lee, Pil H.; Maggiora, Gerald M. Solvation thermodynamics of polar molecules in aqueous solution by the XRISM method. *J. Phys. Chem.*, **1993**, *97*, 10175–10185.

[226] Genheden, S.; Luchko, T.; Gusarov, S.; Kovalenko, A.; Ryde, U. An MM/3D-RISM approach for ligand-binding affinities. *J. Phys. Chem.*, **2010**. Accepted.

[227] Yu, Hsiang-Ai; Roux, Benoit; Karplus, Martin. Solvation thermodynamics: An approach from analytic temperature derivatives. *J. Chem. Phys.*, **1990**, *92*, 5020–5033.

[228] Johnson, J; Case, D. A.; Yamazaki, Takeshi; Gusarov, Sergey; Kovalenko, Andriy; Luchko, Tyler. Small molecule solvation energy and entropy from 3D-RISM. *J. Phys. Condens. Mat.*, **2016**.

[229] Yamazaki, Takeshi; Blinov, Nikolay; Wishart, David; Kovalenko, Andriy. Hydration effects on the HET-s prion and amyloid-$\beta 2$ fibrillous aggregates, studied with three-dimensional molecular theory of solvation. *Biophys. J.*, **2008**, *95*, 4540–4548.

[230] Yamazaki, T.; Kovalenko, A.; Murashov, V.V.; Patey, G.N. Ion solvation in a water-urea mixture. *J. Phys. Chem. B*, **2010**, *114*, 613–619.

[231] Palmer, David S; Frolov, Andrey I; Ratkova, Ekaterina L; Fedorov, Maxim V. Towards a universal method for calculating hydration free energies: a 3D reference interaction site model with partial molar volume correction. *J. Phys.: Condens. Matter*, **2010**, *22*, 492101.

[232] Truchon, Jean-FranÃ§ois; Pettitt, B. Montgomery; Labute, Paul. A cavity corrected 3D-RISM functional for accurate solvation free energies. *J. Chem. Theory Comput.*, **2014**, *10*, 934–941.

[233] Sergiievskyi, Volodymyr; Jeanmairet, Guillaume; Levesque, Maximilien; Borgis, Daniel. Solvation free-energy pressure corrections in the three dimensional reference interaction site model. *J. Chem. Phys.*, **2015**, *143*, 184116.

[234] Luchko, T.; Blinov, N.; Linon, G.C.; Joyce, K.P.; Kovalenko, A. SAMPL5: 3D-RISM partition coefficient calculations with partial molar volume corrections and solute conformational sampling. *J. Comput. Aided Mol. Design*, **2016**, *30*, 1115–1127.

[235] Omelyan, Igor; Kovalenko, Andriy. MTS-MD of biomolecules steered with 3D-RISM-KH mean solvation forces accelerated with generalized solvation force extrapolation. *J. Chem. Theory Comput.*, **2015**, *11*, 1875–1895.

[236] Tuckerman, M.E.; Berne, B.J.; Martyna, G.J. Molecular dynamics algorithm for multiple time scales: Systems with long range forces. *J. Chem. Phys.*, **1991**, *94*, 6811–6815.

[237] Tuckerman, M.; Berne, B.J.; Martyna, G.J. Reversible multiple time scale molecular dynamics. *J. Chem. Phys.*, **1992**, *97*, 1990–2001.

[238] Grubmuumlller, H.; Heller, H.; Windemuth, A.; Schulten, K. Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions. *Mol. Simulat.*, **1991**, *6*, 121–142.

[239] Schlick, T. *Molecular modeling and simulation: an interdisciplinary guide*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[240] Omelyan, Igor; Kovalenko, Andriy. Multiple time step molecular dynamics in the optimized isokinetic ensemble steered with the molecular theory of solvation: Accelerating with advanced extrapolation of effective solvation forces. *J. Chem. Phys.*, **2013**, *139*, 244106.

[241] Omelyan, I.; Kovalenko, A. MTS-MD of Biomolecules Steered with 3D-RISM-KH Mean Solvation Forces Accelerated with Generalized Solvation Force Extrapolation. *J. Chem. Theor. and Comp.*, **2014**, *11*, 1875–1895.

[242] Warshel, A. *Computer Modeling of Chemical Reactions in Enzymes and Solutions*. John Wiley and Sons, New York, 1991.

[243] Billeter, S.R.; Webb, S.P.; Iordanov, T.; Agarwal, P.K.; Hammes-Schiffer, S. Hybrid approach for including electronic and nuclear quantum effects in molecular dynamics simulations of hydrogen transfer reactions in enzymes. *J. Chem. Phys.*, **2001**, *114*, 6925.

[244] Schlegel, H.B.; Sonnenberg, J.L. Empirical valence-bond models for reactive potential energy surfaces using distributed Gaussians. *J. Chem. Theory Comput.*, **2006**, *2*, 905.

[245] Sonnenberg, J.L.; Schlegel, H.B. Empirical valence bond models for reactive potential energy surfaces. II. Intramolecular proton transfer in pyridone and the Claisen reaction of allyl vinyl ether. *Mol. Phys.*, **2007**, *105*, 2719.

[246] Saad, Y.; Schultz, M.H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, **1986**, *7*, 856.

[247] Pulay, P. Convergence acceleration of iterative sequencies. The case of SCF iteration. *Chem. Phys. Lett.*, **1980**, *73*, 393.

[248] Pulay, P. Improved SCF convergence acceleration. *J. Comput. Chem.*, **1982**, *3*, 556.

[249] Feynman, R.P.; Hibbs, A.R. *Quantum Mechanics and Path Integrals*. McGraw-Hill, New York, 1965.

[250] Feynman, R.P. *Statistcal Mechanics*. Benjamin, Reading, MA, 1972.

[251] Kleinert, H. *Path Integrals in Quantum Mechanics, Statistics, and Polymer Physics*. World Scientific, Singapore, 1995.

[252] Kumar, S.; Bouzida, D.; Swendsen, R.H.; Kollman, P.A.; Rosenberg, J.M. The weighted histogram analysis method for free-energy calculations on biomolecules. I. The method. *J. Comput. Chem.*, **1992**, *13*, 1011–1021.

[253] Kumar, S.; Rosenberg, J.M.; Bouzida, D.; Swendsen, R.H.; Kollman, P.A. Multidimensional free-energy calculations using the weighted histogram analysis method. *J. Comput. Chem.*, **1995**, *16*, 1339–1350.

[254] Roux, B. The calculation of the potential of mean force using computer simulations. *Comput. Phys. Comm.*, **1995**, *91*, 275–282.

[255] Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Scalmani, G.; Barone, V.; Mennucci, B.; Petersson, G. A.; Nakatsuji, H.; Caricato, M.; Li, X.; Hratchian, H. P.; Izmaylov, A. F.; Bloino, J.; Zheng, G.; Sonnenberg, J. L.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Vreven, T.; Montgomery, J. A. Jr.; Peralta, J. E.; Ogliaro, F.; Bearpark, M.; Heyd, J. J.; Brothers, E.; Kudin, K. N.; Staroverov, V. N.; Kobayashi, R.; Normand, J.; Raghavachari, K.; Rendell, A.; Burant, J. C.; Iyengar, S. S.; Tomasi, J.; Cossi, M.; Rega, N.; Millam, J. M.; Klene, M.; Knox, J. E.; Cross, J. B.; Bakken, V.; Adamo, C.; Jaramillo, J.; Gomperts, R.; Stratmann, R. E.; Yazyev, O.; Austin, A. J.; Cammi, R.; Pomelli, C.; Ochterski, J. W.; Martin, R. L.; Morokuma, K.; Zakrzewski, V. G.; Voth, G. A.; Salvador, P.; Dannenberg, J. J.; Dapprich, S.; Daniels, A. D.; Farkas, Ö.; Foresman, J. B.; Ortiz, J. V.; Cioslowski, J.; Fox, D. J. Gaussian 09 Revision A.1. Gaussian Inc. Wallingford CT 2009.

[256] Rappe, A.K.; Casewit, C.J.; Colwell, K.S.; Goddard III, W.A.; Skiff, W.M. UFF, a Full Periodic Table Force Field for Molecular Mechanics and Molecular Dynamics Simulations. *J. Am. Chem. Soc.*, **1992**, *114*, 10024–10035.

[257] Walker, R.C.; Crowley, M.F.; Case, D.A. The implementation of a fast and accurate QM/MM potential method in Amber. *J. Comput. Chem.*, **2008**, *29*, 1019–1031.

[258] Seabra, G.M.; Walker, R.C.; Elstner, M.; Case, D.A.; Roitberg, A.E. Implementation of the SCC-DFTB Method for Hybrid QM/MM Simulations within the Amber Molecular Dynamics Package. *J. Phys. Chem. A.*, **2007**, *20*, 5655–5664.

[259] Elstner, M.; Porezag, D.; Jungnickel, G.; Elsner, J.; Haugk, M.; Frauenheim, T.; Suhai, S.; Seifert, G. Self-consistent charge density functional tight-binding method for simulation of complex material properties. *Phys. Rev. B*, **1998**, *58*, 7260.

[260] Kruger, T.; Elstner, M.; Schiffels, P.; Frauenheim, T. Validation of the density-functional based tight-binding approximation. *J. Chem. Phys.*, **2005**, *122*, 114110.

[261] Giese, Timothy J.; York, Darrin M. Charge-dependent model for many-body polarization, exchange, and dispersion interactions in hybrid quantum mechanical/molecular mechanical calculations. *J. Chem. Phys.*, **2007**, *127*, 194101–194111.

[262] Stewart, J.J.P. Optimization of parameters for semiempirical methods I. Method. *J. Comput. Chem.*, **1989**, *10*, 209–220.

[263] Dewar, M.J.S.; Zoebisch, E.G.; Healy, E.F.; Stewart, J.J.P. AM1: A new general purpose quantum mechanical molecular model. *J. Am. Chem. Soc.*, **1985**, *107*, 3902–3909.

[264] Rocha, G.B.; Freire, R.O.; Simas, A.M.; Stewart, J.J.P. RM1: A Reparameterization of AM1 for H, C, N, O, P, S, F, Cl, Br and I. *J. Comp. Chem.*, **2006**, *27*, 1101–1111.

[265] Dewar, M.J.S.; Thiel, W. Ground states of molecules. 38. The MNDO method, approximations and parameters. *J. Am. Chem. Soc.*, **1977**, *99*, 4899–4907.

[266] Repasky, M.P.; Chandrasekhar, J.; Jorgensen, W.L. PDDG/PM3 and PDDG/MNDO: Improved semiempirical methods. *J. Comput. Chem.*, **2002**, *23*, 1601–1622.

[267] McNamara, J.P.; Muslim, A.M.; Abdel-Aal, H.; Wang, H.; Mohr, M.; Hillier, I.H.; Bryce, R.A. Towards a quantum mechanical force field for carbohydrates: A reparameterized semiempirical MO approach. *Chem. Phys. Lett.*, **2004**, *394*, 429–436.

[268] Bernal-Uruchurtu, M. I.; Ruiz-López, M. F. Basic ideas for the correction of semiempirical methods describing H-bonded systems. *Chem. Phys. Lett.*, **2000**, *330*, 118–124.

[269] Arillo-Flores, O. I.; Ruiz-López, M. F.; Bernal-Uruchurtu, M. I. Can semi-empirical models describe HCl dissociation in water? *Theoret. Chem. Acc.*, **2007**, *118*, 425–435.

[270] Thiel, Walter; Voityuk, Alexander A. Extension of the MNDO formalism to d orbitals: Integral approximations and preliminary numerical results. *Theoret. Chim. Acta*, **1992**, *81*, 391–404.

[271] Thiel, Walter; Voityuk, Alexander A. Extension of the MNDO formalism to d orbitals: Integral approximations and preliminary numerical results. *Theoret. Chim. Acta*, **1996**, *93*, 315.

[272] Thiel, Walter; Voityuk, Alexander A. Erratum: Extension of MNDO to d orbitals: Parameters and results for the second-row elements and for the zinc group. *J. Phys. Chem.*, **1996**, *100*, 616–626.

[273] Imhof, Petra; Noé, Frank; Fischer, Stefan; Smith, Jeremy C. AM1/d Parameters for Magnesium in Metalloenzymes. *J. Chem. Theory Comput.*, **2006**, *2*, 1050–1056.

[274] Nam, Kwangho; Cui, Qiang; Gao, Jiali; York, Darrin M. Specific Reaction Parametrization of the AM1/d Hamiltonian for Phosphoryl Transfer Reactions: H, O, and P Atoms. *J. Chem. Theory Comput.*, **2007**, *3*, 486–504.

[275] Stewart, J.J.P. Optimization of parameters for semiempirical methods V: Modification of NDDO approximations and application to 70 elements. *J. Mol. Mod.*, **2007**, *13*, 1173–1213.

[276] Porezag, D.; Frauenheim, T.; Kohler, T.; Seifert, G.; Kaschner, R. Construction of tight-binding-like potentials on the basis of density-functional-theory: Applications to carbon. *Phys. Rev. B*, **1995**, *51*, 12947.

[277] Seifert, G.; Porezag, D.; Frauenheim, T. Calculations of molecules, clusters and solids with a simplified LCAO-DFT-LDA scheme. *Int. J. Quantum Chem.*, **1996**, *58*, 185.

[278] Gaus, Michael; Cui, Qiang; Elstner, Marcus. DFTB3: Extension of the self-consistent-charge density-functional tight-binding method (SCC-DFTB). *J. Chem. Theory Comput.*, **2011**, *7*, 931–948.

[279] Elstner, M.; Hobza, P.; Frauenheim, T.; Suhai, S.; Kaxiras, E. Hydrogen bonding and stacking interactions of nucleic acid base pairs: a density-functional-theory based treatment. *J. Chem. Phys.*, **2001**, *114*, 5149.

[280] Kalinowski, J.A.; Lesyng, B.; Thompson, J.D.; Cramer, C.J.; Truhlar, D.G. Class IV charge model for the self-consistent charge density-functional tight-binding method. *J. Phys. Chem. A*, **2004**, *108*, 2545–2549.

[281] Yang, Y.; Yu, H.; York, D.M.; Cui, Q.; Elstner, M. Extension of the self-consistent charge density-functional tight-binding method: Third-order expansion of the density functional theory total energy and introduction of a modified effective Coulomb interaction. *J. Phys. Chem. A*, **2007**, *111*, 10861–10873.

[282] Korth, Martin. Third-generation hydrogen-bonding corrections for semiempirical qm methods and force fields. *J. Chem. Theory Comput.*, **2010**, *6*, 3808.

[283] Jurecka, Petr; Cerný, Jirí; Hobza, Pavel; Salahub, Dennis R. Density functional theory augmented with an empirical dispersion term. Interaction energies and geometries of 80 noncovalent complexes compared with ab initio quantum mechanics calculations. *J. Comp. Chem.*, **2007**, *28*, 555–569.

[284] Bondi, A. van der Waals volumes and radii. *J. Phys. Chem.*, **1964**, *68*, 441–451.

[285] Korth, Martin; Pitonak, Michal; Rezac, Jan; Hobza, Pavel. A transferable h-bonding correction for semiempirical quantum-chemical methods. *J. Chem. Theory Comput.*, **2010**, *6*, 344–352.

[286] Pellegrini, E.; J. Field, M. A generalized-Born solvation model for macromolecular hybrid-potential calculations. *J. Phys. Chem. A.*, **2002**, *106*, 1316–1326.

[287] Nam, K.; Gao, J.; York, D. An efficient linear-scaling Ewald method for long-range electrostatic interactions in combined QM/MM calculations. *J. Chem. Theory Comput.*, **2005**, *1*, 2–13.

[288] Wang, Q.T.; Bryce, R.A. Improved hydrogen bonding at the NDDO-type semiempirical quantum mechanical/molecular mechanical interface. *J. Chem. Theory Comput.*, **2009**, *5*, 2206–2211.

[289] Götz, Andreas W.; Clark, Matthew A.; Walker, Ross C. An extensible interface for QM/MM molecular dynamics simulations with AMBER. *J. Comput. Chem.*, **2014**, *35*, 95–108.

[290] te Velde, G.; Bickelhaupt, F. M.; Baerends, E. J.; Guerra, C. Fonseca; van Gisbergen, S. J. A.; Snijders, J. G.; Ziegler, T. Chemistry with ADF. *J. Comp. Chem.*, **2001**, *22*, 931–967.

[291] ADF2011, SCM, Theoretical Chemistry, Vrije Universiteit, Amsterdam, The Netherlands, http://www.scm.com, accessed 02/29/2012.

[292] Schmidt, Michael W.; Baldridge, Kim K.; Boatz, Jerry A.; Elbert, Steven T.; Gordon, Mark S.; Jensen, Jan H.; Koseki, Shiro; Matsunaga, Nikita; Nguyen, Kiet A.; Su, Shujun; Windus, Theresa L.; Dupuis, Michel; Montgomery, John A. Jr. General atomic and molecular electronic structure system. *J. Comp. Chem.*, **1993**, *14*, 1347–1363.

[293] Gordon, M. S.; Schmidt, M. W. in *Theory and Applications of Computational Chemistry, the first forty years*, Dykstra, C. E.; Frenking, G.; Kim, K. S.; Scuseria, G. E., Eds., chapter 41, pp 1167–1189. Elsevier, Asterdam, 2005.

[294] Valiev, M.; Bylaska, E. J.; Govind, N.; Kowalski, K.; Straatsma, T. P.; van Dam, H. J. J.; Wang, D.; Niepolcha, J.; Apra, E.; Windus, T. L.; de Jong, W. A. Nwchem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Comput. Phys. Commun.*, **2010**, *181*, 1477.

[295] Neese, F. ORCA - an ab initio, Density Functional and Semiempirical program package, Version 2.8.0, University of Bonn, 2010.

[296] Shao, Yihan; Fusti-Molnar, Laszlo; Jung, Yousung; Kussmann, Jürg; Ochsenfeld, Christian; Brown, Shawn T.; Gilbert, Andrew T. B.; Slipchenko, Lyudmila V.; Levchenko, Sergey V.; O'Neill, Darragh P.; DiStasio, Robert A. Jr.; Lochan, Rohini C.; Wang, Tao; Beran, Gregory J.O.; Besley, Nicholas A.; Herbert, John M.; Lin, Ching Yeh; Voorhis, Troy Van; Chien, Siu Hung; Sodt, Alex; Steele, Ryan P.; Rassolov, Vitaly A.; Maslen, Paul E.; Korambath, Prakashan P.; Adamson, Ross D.; Austin, Brian; Baker, Jon; Byrd, Edward F. C.; Daschel, Holger; Doerksen, Robert J.; Dreuw, Andreas; Dunietz, Barry D.; Dutoi, Anthony D.; Furlani, Thomas R.; Gwaltney, Steven R.; Heyden, Andreas; Hirata, So; Hsu, Chao-Ping; Kedziora, Gary; Khaliullin, Rustam Z.; Klunzinger, Phil; Lee, Aaron M.; Lee, Michael S.; Liang, WanZhen; Lotan, Itay; Nair, Nikhil; Peters, Baron; Proynov, Emil I.; Pieniazek, Piotr A.; Rhee, Young Min; Ritchie, Jim; Rosta, Edina; Sherrill, C. David; Simmonett, Andrew C.; Subotnik, Joseph E.; Woodcock, H. Lee III; Zhang, Weimin; Bell, Alexis T.; Chakraborty, Arup K.; Chipman, Daniel M.; Keil, Frerich J.; Warshel, Arieh; Hehre, Warren J.; Schaefer, Henry F. III; Kong, Jing; Krylov, Anna I.; Gill, Peter M. W.; Head-Gordon, Martin. Advances in methods and algorithms in a modern quantum chemistry program package. *Phys. Chem. Chem. Phys.*, **2006**, *8*, 3172–3191.

[297] Ufimtsev, Ivan S.; Martinez, Todd J. Quantum chemistry on graphical processing units. 3. Analytical energy gradients, geometry optimization, and first principles molecular dynamics. *J. Chem. Theory Comput.*, **2009**, *5*, 2619–2628.

[298] Kállay, M.; Rolik, Z.; Csontos, J.; Ladjánszki, I.; Szegedy, L.; Ladoćzki, B.; Samu, G.; Petrov, K.; Farkas, M.; Nagy, P.; Mester, D.; Hegely, B. Mrcc, a quantum chemical program suite. www.mrcc.hu.

[299] Rolik, Z.; Szegedy, L.; Ladjánszki, I.; Ladóczki, B.; Kállay, M. An efficient linear-scaling CCSD(T) method based on local natural orbitals. *J. Chem. Phys.*, **2013**, *139*, 094105.

[300] Torras, J.; He, Y.; Cao, C.; Muralidharan, K.; Deumens, E.; Cheng, H.; Trickey, S. PUPIL: A systematic approach to software integration in multi-scale simulations. *Comput. Phys. Comm.*, **2007**, *177*, 265–279.

[301] Torras, J.; Seabra, G.; Deumens, E.; Trickey, S.B.; Roitberg, A.E. A versatile AMBER-Gaussian QM/MM interface through PUPIL. *J. Comput. Chem.*, **2008**, *29*, 1564–1573.

[302] Hégely, B.; Nagy, P.R.; Ferenczy, G.G.; Kállay, M. Exact density functional and wave function embedding schemes based on orbital localization. *J. Chem. Phys.*, **2016**, *145*, 064107.

[303] Bulo, Rosa E.; Michel, Carine; Fleurat-Lessard, Paul; Sautet, Philippe. Multiscale modeling of chemistry in water: Are we there yet? *J. Chem. Theory Comput.*, **2013**, *9*, 5567–5577.

[304] Bulo, Rosa E.; Ensing, Bernd; Sikkema, Jetze; Visscher, Lucas. Toward a practical method for adaptive qm/mm simulations. *J. Chem. Theory Comput.*, **2009**, *9*, 2212–2221.

[305] Park, Kyoyeon; Götz, Andreas W.; Walker, Ross C.; Paesani, Francesco. Application of adaptive qm/mm methods to molecular dynamics simulations of aqueous systems. *J. Chem. Theory Comput.*, **2012**, *8*, 2868–2877.

[306] Bernstein, N.; Várnai, C.; Solt, I.; Winfield, S. A.; Payne, M. C.; Simon, I.; Fuxreiter, M.; Csányi, G. *Phys. Chem. Chem. Phys.*, **2011**, *14*, 646–656.

[307] Várnai, Csilla; Bernstein, Noam; Mones, Letif; Csányi, Gábor. Tests of an adaptive qm/mm calculation on free energy profiles of chemical reactions in solution. *J. Phys. Chem. B*, **2013**, *117*, 12202–12211.

[308] Csányi, Gabor; Albaret, T; Moras, G; Payne, M C; Vita, A De. Multiscale hybrid simulation methods for material systems. *J. Phys. Condens. Matt.*, **2005**, *17*, R691.

[309] Bernstein, N.; Kermode, J. R.; Csányi, G. Hybrid atomistic simulation methods for materials systems. *Rep. Prog. Phys.*, **2009**, *72*, 026501.

[310] Jones, Andrew; Leimkuhler, Ben. Adaptive stochastic methods for sampling driven molecular systems. *J. Chem. Phys.*, **2011**, *135*, 084125–084125–11.

[311] Kerdcharoen, T; Rode, Bernd Michael. A QM/MM simulation method applied to the solution of Li+ in liquid ammonia. *Chem. Phys.*, **1996**, *211*, 313–323.

[312] Dixon, S.L.; Merz, K.M. Jr. Semiempirical molecular orbital calculations with linear system size scaling. *J. Chem. Phys.*, **1996**, *104*, 6643–6649.

[313] Dixon, S.L.; Merz, K.M. Jr. Fast, accurate semiempirical molecular orbital calculations for macromolecules. *J. Chem. Phys.*, **1997**, *107*, 879–893.

[314] Monard, G.; Bernal-Uruchurtu, M. I.; Van Der Vaart, A.; Merz, K. M. Jr.; Ruiz-López, M. F. Simulation of liquid water using semiempirical Hamiltonians and the divide and conquer approach. *J. Phys. Chem. A*, **2005**, *109*, 3425–3432.

[315] Marion, A.; Monard, G.; Ruiz-López, M. F.; Ingrosso, F. Water interactions with hydrophobic groups: assessment and recalibration of semiempirical molecular orbital methods. *J. Chem. Phys.*, **2014**, *141*, 034106.

[316] Bernal-Uruchurtu, M. I.; Martins-costa, M. T. C.; Millot, C.; Ruiz-López, M. F. Improving Description of Hydrogen Bonds at the Semiempirical Level : Water - Water Interactions as Test Case. *J. Comput. Chem.*, **2000**, *21*, 572–581.

[317] Harb, W.; Bernal-Uruchurtu, M. I.; Ruiz-López, M. F. An improved semiempirical method for hydrated systems. *Theor. Chem. Acc.*, **2004**, *112*, 204–216.

[318] Thiriot, E.; Monard, G. Combining a genetic algorithm with a linear scaling semiempirical method for protein-ligand docking. *J. Mol. Struct. Theochem*, **2009**, *898*, 31–41.

[319] Ludwig, O.; Schinke, H.; Brandt, W. Reparametrisation of Force Constants in MOPAC 6.0/7.0 for Better Description of the Activation Barrier of Peptide Bond Rotations. *J. Molec. Model.*, **1996**, *2*, 341–350.

[320] MacKerell Jr., A.D.; Bashford, D.; Bellott, M.; Dunbrack, R.L.; Evanseck, J.D.; Field, M.J.; Fischer, S.; Gao, J.; Guo, H.; Ha, S.; Joseph-McCarthy, D.; Kuchnir, L.; Kuczera, K.; Lau, F.T.K.; Mattos, C.; Michnick, S.; Ngo, T.; Nguyen, D.T.; Prodhom, B.; Reiher, W.E.; Roux, B.; Schlenkrich, M.; Smith, J.C.; Stote, R.; Straub, J.; Watanabe, M.; Wiorkiewicz-Kuczera, J.; Yin, D.; Karplus, M. All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins. *J. Phys. Chem. B*, **1998**, *102*, 3586–3616.

[321] MacKerell Jr., A.D.; Banavali, N.; Foloppe, N. Development and current status of the CHARMM force field for nucleic acids. *Biopolymers*, **2000**, *56*, 257–265.

[322] MacKerell, A.D. Jr.; Feig, M.; Brooks III, C.L. Improved Treatment of the Protein Backbone in Empirical Force Fields. *J. Am. Chem. Soc.*, **2004**, *126*, 698–699.

[323] MacKerell, A.D. Jr.; Feig, M.; Brooks III, C.L. Extending the treatment of backbone energetics in protein force fields: Limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations. *J. Computat. Chem.*, **2004**, *25*, 1400–1415.

[324] Hopkins, C.W.; Le Grand, S.; Walker, R.C.; Roitberg, A.E. Long-Time-Step Molecular Dynamics through Hydrogen Mass Repartitioning. *J. Chem. Theory Comput.*, **2015**, *11*, 1864–1874.

[325] Bergonzo, Christina; Henriksen, Niel M.; Roe, Daniel R.; Swails, Jason M.; Roitberg, Adrian E.; Cheatham III, Thomas E. Multidimensional Replica Exchange Molecular Dynamics Yields a Converged Ensemble of an RNA Tetranucleotide. *J. Chem. Theory Comput.*, **2013**, *10*, 492–499.

[326] Wang, J.; Wolf, R.M.; Caldwell, J.W.; Kollamn, P.A.; Case, D.A. Development and testing of a general Amber force field. *J. Comput. Chem.*, **2004**, *25*, 1157–1174.

[327] Wang, B.; Merz, K.M. Jr. A fast QM/MM (quantum mechanical/molecular mechanical) approach to calculate nuclear magnetic resonance chemical shifts for macromolecules. *J. Chem. Theory Comput.*, **2006**, *2*, 209–215.

[328] Peters, M.B.; Yang, Y.; Wang, B.; Fusti-Molnar, L.; Weaver, M.N.; Merz, K.M. Jr. Structural Survey of Zinc-Containing Proteins and Development of the Zinc AMBER Force Field (ZAFF). *J. Chem. Theor. Comput.*, **2010**, *6*, 2935–2947.

[329] Jakalian, A.; Bush, B.L.; Jack, D.B.; Bayly, C.I. Fast, efficient generation of high-quality atomic charges. AM1-BCC model: I. Method. *J. Comput. Chem.*, **2000**, *21*, 132–146.

[330] Jakalian, A.; Jack, D.B.; Bayly, C.I. Fast, efficient generation of high-quality atomic charges. AM1-BCC model: II. Parameterization and Validation. *J. Comput. Chem.*, **2002**, *23*, 1623–1641.

[331] Bayly, C.I.; Cieplak, P.; Cornell, W.D.; Kollman, P.A. A well-Behaved electrostatic potential based method using charge restraints for determining atom-centered charges: The RESP model. *J. Phys. Chem.*, **1993**, *97*, 10269–10280.

[332] Wang, J.; Kollman, P.A. Automatic parameterization of force field by systematic search and genetic algorithms. *J. Comput. Chem.*, **2001**, *22*, 1219–1228.

[333] Graves, A.P.; Shivakumar, D.M.; Boyce, S.E.; Jacobson, M.P.; Case, D.A.; Shoichet, B.K. Rescoring docking hit lists for model cavity sites: Predictions and experimental testing. *J. Mol. Biol.*, **2008**, *377*, 914–934.

[334] Jojart, B.; Martinek, T.A. Performance of the general amber force field in modeling aqueous POPC membrane bilayers. *J. Comput. Chem.*, **2007**, *28*, 2051–2058.

[335] Rosso, L.; Gould, I.R. Structure and dynamics of phospholipid bilayers using recently developed general all-atom force fields. *J. Comput. Chem.*, **2008**, *29*, 24–37.

[336] Wang, J.; Hou, T. Application of Molecular Dynamics Simulations in Molecular Property Prediction. 1. Density and Heat of Vaporization. *J. Chem. Theory Comput.*, **2011**, *7*, 2151–2165.

[337] Mobley, David L.; Bayly, Christopher I.; Cooper, Matthew D.; Shirts, Michael R.; Dill, Ken A. Small Molecule Hydration Free Energies in Explicit Solvent: An Extensive Test of Fixed-Charge Atomistic Simulations. *J. Chem. Theory Comput.*, **2009**, *5*, 350–358.

[338] Li, P.; Merz, K.M. Jr. Metal Ion Modeling Using Classical Mechanics. *Chem. Rev.*, **2017**, *117*, 1564–1686.

[339] Li, P.; Merz, K.M. Jr. MCPB.py: A Python Based Metal Center Parameter Builder. *J. Chem. Inf. Model.*, **2016**, *56*, 599–604.

[340] Eastman, P.; Pande, V.S. OpenMM: A Hardware-Independent Framework for Molecular Simulations. *Computing in Science and Engineering*, **2010**, *12*, 34–39.

[341] Eastman, P.; Friedrichs, M.S.; Chodera, J.D.; Radmer, R.J.; Bruns, C.M.; Ku, J.P.; Beauchamp, K.A.; Lane, T.J.; Wang, L.; Shukla, D.; Tye, T.; Houston, M.; Stich, T.; Klein, C.; Shirts, M.R.; Pande, V.S. OpenMM 4: A Reusable, Extensible, Hardware Independent Library for High Performance Molecular Simulation. *J. Chem. Theory Comput.*, **2013**, *9*, 461–469.

[342] Yu, Z.; Li, P.; Merz, K.M. Jr. The Extended Zinc AMBER Force Field (EZAFF) Based on an Empirical Approach. *Manuscript in preparation*, **2017**.

[343] Seminario, J.M. Calculation of Intramolecular Force Fields from Second-Derivative Tensors. *Int. J. Quantum Chem.*, **1996**, *30*, 1271–1277.

[344] Kopitz, H.; Zivkovic, A.; Engels, J.W.; Gohlke, H. Determinants of the unexpected stability of RNA fluorobenzene self pairs. *ChemBioChem*, **2008**, *9*, 2619–2622.

[345] Morishita, T. Fluctuation formulas in molecular-dynamics simulations with the weak coupling heat bath. *J. Chem. Phys.*, **2000**, *113*, 2976.

[346] Mudi, A.; Chakravarty, C. Effect of the Berendsen thermostat on the dynamical properties of water. *Mol. Phys.*, **2004**, *102*, 681–685.

[347] Berendsen, H.J.C.; Postma, J.P.M.; van Gunsteren, W.F.; DiNola, A.; Haak, J.R. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.*, **1984**, *81*, 3684–3690.

[348] Harvey, S.C.; Tan, R.K.; Cheatham, T.E. III. The flying ice cube: Velocity rescaling in molecular dynamics leads to violation of energy equipartition. *J. Comput. Chem.*, **1998**, *19*, 726–740.

[349] Andrea, T.A.; Swope, W.C.; Andersen, H.C. The role of long ranged forces in determining the structure and properties of liquid water. *J. Chem. Phys.*, **1983**, *79*, 4576–4584.

[350] Andersen, H.C. Molecular dynamics simulations at constant pressure and/or temperature. *J. Chem. Phys.*, **1980**, *72*, 2384–2393.

[351] Uberuaga, B.P.; Anghel, M.; Voter, A.F. Synchronization of trajectories in canonical molecular-dynamics simulations: Observation, explanation, and exploitation. *J. Chem. Phys.*, **2004**, *120*, 6363–6374.

[352] Sindhikara, D.J.; Kim, S.; Voter, A.F.; Roitberg, A.E. Bad seeds sprout perilous dynamics: Stochastic thermostat induced trajectory synchronization in biomolecules. *J. Chem. Theory Comput.*, **2009**, *5*, 1624–1631.

[353] Omelyan, Igor; Kovalenko, Andriy. Generalized canonical-isokinetic ensemble: Speeding up multiscale molecular dynamics and coupling with 3d molecular theory of solvation. *Mol. Sim.*, **2013**, *39*, 25–48.

[354] Leimkuhler, B.; Margul, D.T.; Tuckerman, M.E. Stochastic, Resonance-Free Multiple Time-Step Algorithm for Molecular Dynamics with Very Large Time Steps. *Mol. Phys.*, **2013**, *111*, 3579–3594.

[355] Pastor, R.W.; Brooks, B.R.; Szabo, A. An analysis of the accuracy of Langevin and molecular dynamics algorithms. *Mol. Phys.*, **1988**, *65*, 1409–1419.

[356] Loncharich, R.J.; Brooks, B.R.; Pastor, R.W. Langevin dynamics of peptides: The frictional dependence of isomerization rates of N-actylananyl-N'-methylamide. *Biopolymers*, **1992**, *32*, 523–535.

[357] Rhee, Young Min; Pande, Vijay S. Solvent viscosity dependence of the protein folding dynamics. *J. Phys. Chem. B*, **2008**, *112*, 6221–6227. PMID: 18229911.

[358] Izaguirre, J.A.; Catarello, D.P.; Wozniak, J.M.; Skeel, R.D. Langevin stabilization of molecular dynamics. *J. Chem. Phys.*, **2001**, *114*, 2090–2098.

[359] Zhang, Y.; Feller, S.E.; Brooks, B.R.; Pastor, R.W. Computer simulation of liquid/liquid interfaces. I. Theory and application to octane/water. *J. Chem. Phys.*, **1995**, *103*, 10252–10266.

[360] Ryckaert, J.-P.; Ciccotti, G.; Berendsen, H.J.C. Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes. *J. Comput. Phys.*, **1977**, *23*, 327–341.

[361] Miyamoto, S.; Kollman, P.A. SETTLE: An analytical version of the SHAKE and RATTLE algorithm for rigid water models. *J. Comput. Chem.*, **1992**, *13*, 952–962.

[362] Wu, X.; Subramaniam, S.; Case, D.A.; Wu, K.; Brooks, B.R. Targeted conformational search with map-restrained self-guided langevin dynamics: application to flexible fitting into electron microscopic density maps. *J. Struct. Biology*, **2013**, *183*, 429–440.

[363] Ren, P.; Ponder, J.W. Consistent treatment of inter- and intramolecular polarization in molecular mechanics calculations. *J. Comput. Chem.*, **2002**, *23*, 1497–1506.

[364] Ren, P.; Ponder, J.W. Temperature and pressure dependence of the AMOEBA water model. *J. Phys. Chem. B*, **2004**, *108*, 13427–13437.

[365] Darden, T.; York, D.; Pedersen, L. Particle mesh Ewald–an Nlog(N) method for Ewald sums in large systems. *J. Chem. Phys.*, **1993**, *98*, 10089–10092.

[366] Essmann, U.; Perera, L.; Berkowitz, M.L.; Darden, T.; Lee, H.; Pedersen, L.G. A smooth particle mesh Ewald method. *J. Chem. Phys.*, **1995**, *103*, 8577–8593.

[367] Crowley, M.F.; Darden, T.A.; Cheatham, T.E. III; Deerfield, D.W. II. Adventures in improving the scaling and accuracy of a parallel molecular dynamics program. *J. Supercomput.*, **1997**, *11*, 255–278.

[368] Sagui, C.; Darden, T.A. in *Simulation and Theory of Electrostatic Interactions in Solution*, Pratt, L.R.; Hummer, G., Eds., pp 104–113. American Institute of Physics, Melville, NY, 1999.

[369] Toukmaji, A.; Sagui, C.; Board, J.; Darden, T. Efficient particle-mesh Ewald based approach to fixed and induced dipolar interactions. *J. Chem. Phys.*, **2000**, *113*, 10913–10927.

[370] Sagui, C.; Pedersen, L.G.; Darden, T.A. Towards an accurate representation of electrostatics in classical force fields: Efficient implementation of multipolar interactions in biomolecular simulations. *J. Chem. Phys.*, **2004**, *120*, 73–87.

[371] Wu, X.; Brooks, B.R. Isotropic periodic sum: A method for the calculation of long-range interactions. *J. Chem. Phys.*, **2005**, *122*, 044107.

[372] Klauda, J.B.; Wu, X.; Pastor, R.W.; Brooks, B.R. Long-Range Lennard-Jones and Electrostatic Interactions in Interfaces. *J. Phys. Chem. B*, **2007**, *111*, 4393–4400.

[373] Takahashi, K.; Yasuoka, K.; Narumi, T. Cutoff radius effect of isotropic periodic sum method for transport. *J. Chem. Phys.*, **2007**, *127*, 114511.

[374] Wu, X.; Brooks, B.R. Using the Isotropic Periodic Sum Method to Calculate Long-Range Interactions of Heterogeneous Systems. *J. Chem. Phys.*, **2008**, *129*, 154115.

[375] Wu, X.; Brooks, B.R. Isotropic periodic sum of electrostatic interactions for polar systems. *J. Chem. Phys.*, **2009**, *131*, 024107.

[376] Venable, R.M.; Chen, L.E.; Pastor, R.W. Comparison of the Extended Isotropic Periodic Sum and Particle Mesh Ewald Methods for Simulations of Lipid Bilayers and Monolayers. *J. Phys. Chem. B*, **2009**, *113*, 5855–5862.

[377] Konecny, Robert; Baker, Nathan A.; McCammon, J. Andrew. iAPBS: a programming interface to the adaptive Poisson–Boltzmann solver. *Comput. Sci. Disc.*, **2012**, *5*, 15005–15013.

[378] Goetz, Andreas W; Williamson, Mark J; Xu, Dong; Poole, Duncan; Grand, Scott Le; Walker, Ross C. Routine microsecond molecular dynamics simulations with AMBER - Part I: Generalized Born. *J. Chem. Theory Comput.*, **2012**, *8*, 1542–1555.

[379] Salomon-Ferrer, Romelia; Goetz, Andreas W; Poole, Duncan; Grand, Scott Le; Walker, Ross C. Routine microsecond molecular dynamics simulations with AMBER - Part 2: Explicit Solvent Particle Mesh Ewald . *J. Chem. Theory Comput.*, **2012**, *in review*.

[380] Le Grand, S.; GÃ¶tz, A.W.; Walker, R.C. SPFP: Speed without compromise–A mixed precision model for GPU accelerated molecular dynamics simulations. *Comput. Phys. Commun.*, **2013**, *184*, 374–380.

[381] Betz, R.M.; DeBardeleben, N.A.; Walker, R.C. An Investigation of the effects of hard and soft errors on graphics processing unit-accelerated molecular dynamics simulations. *Concurrency and Computation: Practice and Experience*, **2014**, *26*, 2134.

[382] Wu, X.; Brooks, B.R. Self-guided Langevin dynamics simulation method. *Chem. Phys. Lett.*, **2003**, *381*, 512–518.

[383] Wu, X.; Damjanovic, A.; Brooks, B. R. Efficient and unbiased sampling of biomolecular systems in the canonical ensemble: a review of self-guided langevin dynamics. *Adv. Chem. Phys.*, **2012**, *150*, 255–326.

[384] Yu, X.; Wu, X.; Bermejo, G.A.; Brooks, B.R.; Taraska, J.W. Accurate high-throughput structure mapping and prediction with transition metal ion fret. *Structure*, **2013**, *21*, 9–19.

[385] Wu, X.; Brooks, B. R. Self-guided langevin dynamics via generalized langevin equation. *J. Comput. Chem.*, **2016**, *37*, 595–601.

[386] Wu, X.; Brooks, B.R. Toward canonical ensemble distribution from self-guided Langevin dynamics simulation. *J. Chem. Phys.*, **2011**, *134*, 134108.

[387] Wu, X.; Brooks, B.R. Force-momentum-based self-guided Langevin dynamics: a rapid sampling method that approaches the canonical ensemble . *J. Chem. Phys.*, **2011**, *135*, 204101.

[388] Wu, X.; Hodoscek, M.; Brooks, B.R. Replica exchanging self-guided Langevin dynamics for efficient and accurate conformational sampling. *J. Chem. Phys.*, **2012**, *137*, 044106.

[389] Hamelberg, Donald; Mongan, John; McCammon, J. Andrew. Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules. *J. Chem. Phys.*, **2004**, *120*, 11919–11929.

[390] Hamelberg, Donald; de Oliveira, Cesar Augusto F.; McCammon, J.Andrew. Sampling of slow diffusive conformational transitions with accelerated molecular dynamics. *J. Chem. Phys.*, **2007**, *127*, 155102–155109.

[391] Grant, Barry J.; Gorfe, Alemayehu A.; McCammon, J. Andrew. Ras conformational switching: Simulating nucleotide-dependent conformational transitions with accelerated molecular dynamics. *PLoS Computat. Biol.*, **2009**, *5*, e1000325.

[392] de Oliveira, Cesar Augusto F.; Grant, Barry J.; Zhou, Michelle; McCammon, J. Andrew. Large-scale conformational changes of trypanosoma cruzi proline racemase predicted by accelerated molecular dynamics simulation. *PLoS Computat. Biol.*, **2011**, *7*, e1002178.

[393] Pierce, Levi C.T.; Salomon-Ferrer, Romelia; de Oliveira, Cesar Augusto F; McCammon, J. Andrew; Walker, Ross C. Routine access to milli-second time scales with accelerated molecular dynamics. *J. Chem. Theory Comput.*, **2012**, *8*, 2997–3002.

[394] Doshi, Urmi; Hamelberg, Donald. Reoptimization of the amber forcefield for peptide bond (omega) torsions using accelerated molecular dynamics. *J. Chem. Phys. B*, **2009**, *113*, 16590–16595.

[395] Miao, Y.; Feher, V.A.; McCammon, J.A. Gaussian Accelerated Molecular Dynamics: Unconstrained Enhanced Sampling and Free Energy Calculation. *J. Chem. Theory Comput.*, **2015**, *11*, 3584–3595.

[396] Miao, Y.; Sinko, W.; Pierce, L.; Bucher, D.; Walker, R.C.; McCammon, J.A. Improved Reweighting of Accelerated Molecular Dynamics Simulations for Free Energy Calculation. *J. Chem. Theory Comput.*, **2014**, *10*, 2677–2689.

[397] Mills, G.; Jönsson, H. Quantum and thermal effects in H_2 dissociative adsorption: Evaluation of free energy barriers in multidimensional quantum systems. *Phys. Rev. Lett.*, **1994**, *72*, 1124–1127.

[398] Jönsson, H.; Mills, G.; Jacobsen, K.W. in *Classical and Quantum Dynamics in Condensed Phase Simulations*, Berne, B.J.; Ciccoti, G.; Coker, D.F., Eds., pp 385–404. World Scientific, Singapore, 1998.

[399] Elber, R.; Karplus M, M. A method for determining reaction paths in large molecules: Application to myoglobin. *Chem. Phys. Lett.*, **1987**, *139*, 375–380.

[400] Henkelman, G.; Jönsson, H. Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.*, **2000**, *113*, 9978–9985.

[401] Henkelman, G.; Uberuaga, B.P.; Jönsson, H. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *J. Chem. Phys.*, **2000**, *113*, 9901–9904.

[402] Chu, J.; Trout, B.L.; Brooks, B.R. A super-linear minimization scheme for the nudged elastic band method. *J. Chem. Phys.*, **2003**, *119*, 12708–12717.

[403] Bergonzo, Christina; Campbell, Arthur J; Walker, Ross C; Simmerling, Carlos. A Partial Nudged Elastic Band Implementation for Use with Large or Explicitly Solvated Systems. *Int J Quantum Chem*, **2009**, *109*, 3781–3790.

[404] Mathews, D.H.; Case, D.A. Nudged Elastic Band calculation of minimal energy pathways for the conformational change of a GG mismatch. *J. Mol. Biol.*, **2006**, *357*, 1683–1693.

[405] Kolossváry, I.; Guida, W.C. Low mode search. An efficient, automated computational method for conformational analysis: Application to cyclic and acyclic alkanes and cyclic peptides. *J. Am. Chem. Soc.*, **1996**, *118*, 5011–5019.

[406] Kolossváry, I.; Guida, W.C. Low-mode conformatinoal search elucidated: Application to $C_{39}H_{80}$ and flexible docking of 9-deazaguanine inhibitors into PNP. *J. Comput. Chem.*, **1999**, *20*, 1671–1684.

[407] Kolossváry, I.; Keserü, G.M. Hessian-free low-mode conformational search for large-scale protein loop optimization: Application to c-jun N-terminal kinase JNK3. *J. Comput. Chem.*, **2001**, *22*, 21–30.

[408] Keserü, G.M.; Kolossváry, I. Fully flexible low-mode docking: Application to induced fit in HIV integrase. *J. Am. Chem. Soc.*, **2001**, *123*, 12708–12709.

[409] Press, W.H.; Flannery, B.P.; Teukolsky, S.A.; Vetterling, W.T. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, 1989.

[410] Liu, D.C.; Nocedal, J. On the limited memory method for large scale optimization. *Math. Programming B*, **1989**, *45*, 503–528.

[411] Nocedal, J.; L. Morales, J. Automatic preconditioning by limited memory quasi-Newton updating. *SIAM J. Opt.*, **2000**, *10*, 1079–1096.

[412] Kollman, P. Free energy calculations: Applications to chemical and biochemical phenomena. *Chem. Rev.*, **1993**, *93*, 2395–2417.

[413] Simonson, T. in *Computational Biochemistry and Biophysics*, Becker, O.; MacKerell, A.D.; Roux, B.; Watanabe, M., Eds. Marcel Dekker, New York, 2001.

[414] Steinbrecher, T.; Case, D.A.; Labahn, A. A multistep approach to structure-based drug design: Studying ligand binding at the human neutrophil elastase. *J. Med.. Chem.*, **2006**, *49*, 1837–1844.

[415] Steinbrecher, T.; Hrenn, A.; Dormann, K.; Merfort, I.; Labahn, A. Bornyl (3,4,5-trihydroxy)-cinnamate - An optimized human neutrophil elastase inhibitor designed by free energy calculations. *Bioorg. Med. Chem.*, **2008**, *16*, 2385–2390.

[416] Kaus, J.; Pierce, L.C.T.; Walker, R.C.; McCammon, J.A. PMEMD TI: Placeholder. *J. Chem. Theory Comput.*, **2013**.

[417] Hummer, G.; Szabo, A. Calculation of free-energy differences from computer simulations of initial and final states. *J. Chem. Phys.*, **1996**, *105*, 2004–2010.

[418] Steinbrecher, T.; Mobley, D.L.; Case, D.A. Non-linear scaling schemes for Lennard-Jones interactions in free energy calculations. *J. Chem. Phys.*, **2007**, *127*, 214108.

[419] Steinbrecher, T.; Joung, I.; Case, D.A. Soft-core potentials in thermodynamic integration: Comparing one- and two-step transformations. *J. Comp. Chem.*, **2011**, *32*, 3253–3263.

[420] Schilling, T.; Schmid, F. Computing absolute free energies of disordered structures by molecular simulation. *J. Chem. Phys.*, **2009**, *131*, 231102.

[421] Schmid, Friederike; Schilling, Tanja. A method to compute absolute free energies or enthalpies of fluids. *Physics Procedia*, **2010**, *4*, 131–143.

[422] Berryman, Joshua T.; Schilling, Tanja. Free Energies by Thermodynamic Integration Relative to an Exact Solution, Used to Find the Handedness-Switching Salt Concentration for DNA. *J. Chem. Theory Comput.*, **2013**, *9*, 679–686.

[423] Berryman, Joshua T.; Schilling, Tanja. Absolute Free Energies for Biomolecules in Implicit or Explicit Solvent. *Physics Procedia*, **2014**, *57*, 7–15.

[424] Frenkel, Daan; Ladd, Anthony J. C. New Monte Carlo method to compute the free energy of arbitrary solids. Application to the fcc and hcp phases of hard spheres. *J. Chem. Phys.*, **1984**, *81*, 3188–3193.

[425] Vega, Carlos; Noya, Eva G. Revisiting the Frenkel-Ladd method to compute the free energy of solids: the Einstein molecule approach. *J. Chem. Phys.*, **2007**, *127*, 154113.

[426] Assaraf, Roland; Caffarel, Michel; Kollias, A. C. Chaotic versus nonchaotic stochastic dynamics in monte carlo simulations: A route for accurate energy differences in n-body systems. *Phys. Rev. Lett.*, **2011**, *106*, 150601.

[427] Valleau, J.P.; Torrie, G.M. in *Modern Theoretical Chemistry, Vol. 5: Statistical Mechanics, Part A,*, Berne, B.J., Ed. Plenum Press, New York, 1977.

[428] Kottalam, J.; Case, D.A. Dynamics of ligand escape from the heme pocket of myoglobin. *J. Am. Chem. Soc.*, **1988**, *110*, 7690–7697.

[429] Kästner, J.; Thiel, W. Bridiging the gap between thermodynamic integration and umbrella sampling provides a novel analysis method: "Umbrella integration". *J. Chem. Phys.*, **2005**, *123*, 144104.

[430] Mitsutake, A.; Sugita, Y.; Okamoto, Y. Generalized-ensemble algorithms for molecular simulations of biopolymers. *Biopolymers*, **2001**, *60*, 96–123.

[431] Nymeyer, H.; Gnanakaran, S.; García, A.E. Atomic simulations of protein folding using the replica exchange algorithm. *Meth. Enzymol.*, **2004**, *383*, 119–149.

[432] Cheng, X.; Cui, G.; Hornak, V.; Simmerling, C. Modified replica exchange simulation methods for local structure refinement. *J. Phys. Chem. B*, **2005**, *109*, 8220–8230.

[433] Meng, Yilin; Sabri Dashti, Danial; Roitberg, Adrian E. Computing Alchemical Free Energy Differences with Hamiltonian Replica Exchange Molecular Dynamics (H-REMD) Simulations. *J. Chem. Theory Comput.*, **2011**, *7*, 2721–2727.

[434] Itoh, Satoru G.; Damjanovic, Ana; Brooks, Bernard R. pH replica-exchange method based on discrete protonation states. *Proteins*, **2011**, *79*, 3420–3436.

[435] Swails, Jason M.; Roitberg, Adrian E. Enhancing Conformation and Protonation State Sampling of Hen Egg White Lysozyme Using pH Replica Exchange Molecular Dynamics. *J. Chem. Theory Comput.*, **2012**, *8*, 4393–4404.

[436] Okur, A.; Roe, D.R.; Cui, G.; Hornak, V.; Simmerling, C. Improving convergence of replica-exchange simulations through coupling to a high-temperature structure reservoir. *J. Chem. Theory comput.*, **2007**, *3*, 557–568.

[437] Roitberg, A.E.; Okur, A.; Simmerling, C. Coupling of replica exchange simulations to a non-Boltzmann structure reservoir. *J. Phys. Chem. B*, **2007**, *111*, 2415–2418.

[438] Sabri Dashti, Danial; Roitberg, Adrian E. Calculating the pKa Shift of Titratable Group at Position 66 of Staphylococcal Nuclease Mutant with the Replica Exchange Free Energy Perturbation method (REFEP). *In preparation*, **2012**.

[439] Sabri Dashti, Danial; Roitberg, Adrian E. Optimization of Umbrella Sampling Replica Exchange Molecular Dynamics by Replica Positioning. *J. Chem. Theory Comput.*, **2013**, *9*, 4692–4699.

[440] Fajer, Mikolai; Hamelberg, Donald; McCammon, J. Andrew. Replica-Exchange Accelerated Molecular Dynamics (REXAMD) Applied to Thermodynamic Integration. *J. Chem. Theory Comput.*, **2008**, *4*, 1565–1569.

[441] Arrar, Mehrnoosh; de Oliveira, Cesar Augusto F.; Fajer, Mikolai; Sinko, William; McCammon, J. Andrew. w-REXAMD: A Hamiltonian Replica Exchange Approach to Improve Free Energy Calculations for Systems with Kinetically Trapped Conformations. *J. Chem. Theory Comput.*, **2013**, *9*, 18–23.

[442] Shirts, Michael R; Chodera, John D. Statistically optimal analysis of samples from multiple equilibrium states. *J. Chem. Phys.*, **2008**, *129*, 124105–124105–10.

[443] Pohorille, Andrew; Jarzynski, Christopher; Chipot, Christophe. Good practices in Free-Energy calculations. *J. Phys. Chem. B*, **2010**, *114*, 10235–10253.

[444] Swails, Jason M. *Free Energy Simulations of Complex Biological Systems at Constant pH*. PhD thesis, University of Florida, 2013.

[445] Babin, V.; Roland, C.; Sagui, C. Adaptively biased molecular dynamics for free energy calculations. *J. Chem. Phys.*, **2008**, *128*, 134101.

[446] Huber, Thomas; Torda, Andrew E.; van Gunsteren, Wilfred F. Local elevation: a method for improving the searching properties of molecular dynamics simulation. *J. Comput. Aided. Mol. Des.*, **1994**, *8*, 695–708.

[447] Wang, Fugao; Landau, D. P. Efficient, multiple-range random walk algorithm to calculate the density of states. *Phys. Rev. Lett.*, **2001**, *86*, 2050–2053.

[448] Laio, A.; Parrinello, M. Escaping free-energy minima. *Proc. Natl. Acad. Sci.*, **2002**, *99*, 12562–12566.

[449] Iannuzzi, M.; Laio, A.; Parrinello, M. Efficient exploration of reactive potential energy surfaces using car-parrinello molecular dynamics. *Phys. Rev. Lett.*, **2003**, *90*, 238302–1.

[450] Lelièvre, Tony; Rousset, Mathias; Stoltz, Gabriel. Computation of free energy profiles with parallel adaptive dynamics. *J. Chem. Phys.*, **2007**, *126*, 134111.

[451] Raiteri, Paolo; Laio, Alessandro; Gervasio, Francesco Luigi; Micheletti, Cristian; Parrinello, Michele. Efficient reconstruction of complex free energy landscapes by multiple walkers metadynamics. *J. Phys. Chem.*, **2006**, *110*, 3533–3539.

[452] Sugita, Yuji; Kitao, Akio; Okamoto, Yuko. Multidimensional replica-exchange method for free-energy calculations. *J. Chem. Phys.*, **2000**, *113*, 6042–6051.

[453] Bussi, Giovanni; Gervasio, Francesco Luigi; Laio, Alessandro; Parrinello, Michele. Free-energy landscape for β hairpin folding from combined parallel tempering and metadynamics. *J. Am. Chem. Soc.*, **2006**, *128*, 13435–13441.

[454] Piana, S.; Laio, A. A bias-exchange approach to protein folding. *J. Phys. Chem. B*, **2007**, *111*, 4553–4559.

[455] Babin, V.; Roland, C.; Darden, T. A.; Sagui, C. The free energy landscape of small peptides as obtained from metadynamics with umbrella sampling corrections. *J. Chem. Phys.*, **2006**, *125*, 2049096.

[456] Babin, V.; Karpusenka, V.; Moradi, M.; Roland, C.; Sagui, C. Adaptively biased molecular dynamics: an umbrella sampling method with a time dependent potential. *Inter. J. Quantum Chem.*, **2009**, *109*, 3666–3678.

[457] Babin, Volodymyr; Sagui, Celeste. Conformational free energies of methyl-alpha-l-iduronic and methyl-beta-d-glucuronic acids in water. *J. Chem. Phys.*, **2010**, *132*, 104108.

[458] Moradi, M.; Babin, V.; Roland, C.; Darden, T.; Sagui, C. Conformations and free energy landscapes of polyproline peptides. *Proc. Natl. Aca. Sci. USA*, **2009**, *106*, 20746.

[459] Moradi, M.; Babin, V.; Roland, C.; Sagui, C. A classical molecular dynamics investigation of the free energy and structure of short polyproline conformers. *J. Chem. Phys.*, **2010**, *133*, 125104.

[460] Moradi, M.; Babin, V.; Sagui, C.; Roland, C. in *Proline: Biosynthesis, Regulation and Health Benefits*, B. Nedjimi, Ed., pp 67–110. Nova Publishers, 2013.

[461] Moradi, M; Babin, V.; Sagui, C.; Roland, C. A statistical analysis of the PPII propensity of amino acid guests in proline-rich peptides. *Biophysical J.*, **2011**, *100*, 1083 – 1093.

[462] Moradi, M.; Babin, V.; Sagui, C.; Roland, C. PPII propensity of multiple-guest amino acids in a proline-rich environment. *J. Phys. Chem. B.*, **2011**, *115*, 8645–8656.

[463] Babin, V.; Roland, C.; Sagui, C. The alpha-sheet: A missing-in-action secondary structure? *Proteins –Structure Function and Bioinformatics*, **2011**, *79*, 937–946.

[464] Moradi, M.; Babin, V.; Roland, C.; Sagui, C. Are long-range structural correlations behind the aggregation phenomena of polyglutamine diseases? *PLoS Comput. Biol.*, **2012**, *8*, e1002501.

[465] Moradi, M.; Babin, V.; Roland, C.; Sagui, C. Reaction path ensemble of the B-Z-DNA transition: a comprehensive atomistic study. *Nucleic Acids Research*, **2013**, *41*, 33–43.

[466] Moradi, M.; Lee, J.-G.; Babin, V.; Roland, C.; Sagui, C. Free energy and structure of polyproline peptides: an ab initio and classical molecular dynamics investigation. *Int. J. Quantum Chem.*, **2010**, *110*, 2865–2879.

[467] Moradi, M.; Sagui, C.; Roland, C. Calculating relative transition rates with driven nonequilibrium simulations. *Chem. Phys. Lett.*, **2011**, *518*, 109.

[468] Moradi, M.; Sagui, C.; Roland, C. Investigating rare events with nonequilibrium work measurements: I. Nonequilibrium transition paths. *J. Chem. Phys.*, **2014**, *140*, 034114.

[469] Moradi, M.; Sagui, C.; Roland, C. Investigating rare events with nonequilibrium work measurements: II. Transition and reaction rates. *J. Chem. Phys.*, **2014**, *140*, 034115.

[470] Moradi, M.; Tajkorshid, E. Driven Metadynamics: Reconstructing equilibrium free energies from driven adaptive-bias simulations. *J. Phys. Chem. Lett.*, **2013**, *4*, 1882.

[471] Pan, A.C.; Sezer, D.; Roux, B. Finding transition pathways using the string method with swarms of trajectories. *J. Phys. Chem. B*, **2008**, *112*, 3432–3440.

[472] A. Barducci and G. Bussi and M. Parrinello. Well-tempered metadynamics: a smoothly converging and tunable free energy method. *Phys. Rev. Lett.*, **2008**, *100*, 020603.

[473] K. Minoukadeh and Ch. Chipot and T. Lelievre. Potential of Mean Force Calculations: A multiple-walker adaptive biasing force technique. *J. Chem. Theor. and Comput.*, **2010**, *6*, 1008.

[474] Coutsias, E. A.; Seok, C.; Dill, K. A. Using quaternions to calculate RMSD. *J. Comput. Chem.*, **2004**, *25*, 1849–1857.

[475] Park, Sanghyun; Khalili-Araghi, Fatemeh; Tajkhorshid, Emad; Schulten, Klaus. Free energy calculation from steered molecular dynamics simulations using Jarzynski's equality. *J. Chem. Phys.*, **2003**, *119*, 3559–3566.

[476] Matsumoto, Makoto; Nishimura, Takuji. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, **1998**, *8*, 3–30.

[477] Maragliano, L.; Fischer, A.; Vanden-Eijnden, E.; Ciccotti, G. String method in collective variables: Minimum free energy paths and isocommittor surfaces. *J. Chem. Phys.*, **2006**, *125*, 024106.

[478] Jensen, M.O.; Park, S.; d, E.Tajkhorshi; Schulten, K. Energetics of glycerol conduction through aquaglyceroporin GlpF. *Proc. Natl. Acad. Sci. USA*, **2002**, *99*, 6731–6736.

[479] Crespo, A.; Marti, M.A.; Estrin, D.A.; Roitberg, A.E. Multiple-steering QM-MM calculation of the free energy profile in chorismate mutase. *J. Am. Chem. Soc.*, **2005**, *127*, 6940–6941.

[480] Jarzynski, C. Nonequilibrium equality for free energy differences. *Phys. Rev. Lett.*, **1997**, *78*, 2690–2693.

[481] Hummer, G.; Szabo, A. Free energy reconstruction from nonequilibrium single-molecule pulling experiments. *Proc. Natl. Acad. Sci. USA*, **2001**, *98*, 3658.

[482] Hummer, G.; Szabo, A. Kinetics from nonequilibrium single-molecule pulling experiments. *Biophys. J.*, **2003**, *85*, 5–15.

[483] Mongan, J.; Case, D.A.; McCammon, J.A. Constant pH molecular dynamics in generalized Born implicit solvent. *J. Comput. Chem.*, **2004**, *25*, 2038–2048.

[484] Swails, Jason M.; York, Darrin M.; Roitberg, Adrian E. Constant pH replica exchange molecular dynamics in explicit solvent using discrete protonation states: implementation, testing, and validation. *J. Chem. Theory Comput.*, **2014**, *10*, 1341–1352.

[485] Duggan, B.M.; Legge, G.B.; Dyson, H.J.; Wright, P.E. SANE (Structure Assisted NOE Evaluation): An automated model-based approach for NOE assignment. *J. Biomol. NMR*, **2001**, *19*, 321–329.

[486] Kalk, A.; Berendsen, H.J.C. Proton magnetic relaxation and spin diffusion in proteins. *J. Magn. Reson.*, **1976**, *24*, 343–366.

[487] Olejniczak, E.T.; Weiss, M.A. Are methyl groups relaxation sinks in small proteins? *J. Magn. Reson.*, **1990**, *86*, 148–155.

[488] Cross, K.J.; Wright, P.E. Calibration of ring-current models for the heme ring. *J. Magn. Reson.*, **1985**, *64*, 220–231.

[489] Ösapay, K.; Case, D.A. A new analysis of proton chemical shifts in proteins. *J. Am. Chem. Soc.*, **1991**, *113*, 9436–9444.

[490] Case, D.A. Calibration of ring-current effects in proteins and nucleic acids. *J. Biomol. NMR*, **1995**, *6*, 341–346.

[491] Banci, L.; Bertini, I.; Gori-Savellini, G.; Romagnoli, A.; Turano, P.; Cremonini, M.A.; Luchinat, C.; Gray, H.B. Pseudocontact shifts as constraints for energy minimization and molecular dynamics calculations on solution structures of paramagnetic metalloproteins. *Proteins*, **1997**, *29*, 68.

[492] Sanders, C.R. II; Hare, B.J.; Howard, K.P.; Prestegard, J.H. Magnetically-oriented phospholipid micelles as a tool for the study of membrane-associated molecules. *Prog. NMR Spectr.*, **1994**, *26*, 421–444.

[493] Tsui, V.; Zhu, L.; Huang, T.H.; Wright, P.E.; Case, D.A. Assessment of zinc finger orientations by residual dipolar coupling constants. *J. Biomol. NMR*, **2000**, *16*, 9–21.

[494] Case, D.A. Calculations of NMR dipolar coupling strengths in model peptides. *J. Biomol. NMR*, **1999**, *15*, 95–102.

[495] Gippert, G.P.; Yip, P.F.; Wright, P.E.; Case, D.A. Computational methods for determining protein structures from NMR data. *Biochem. Pharm.*, **1990**, *40*, 15–22.

[496] Case, D.A.; Wright, P.E. in *NMR in Proteins*, Clore, G.M.; Gronenborn, A., Eds., pp 53–91. MacMillan, New York, 1993.

[497] Case, D.A.; Dyson, H.J.; Wright, P.E. Use of chemical shifts and coupling constants in nuclear magnetic resonance structural studies on peptides and proteins. *Meth. Enzymol.*, **1994**, *239*, 392–416.

[498] Brüschweiler, R.; Case, D.A. Characterization of biomolecular structure and dynamics by NMR cross-relaxation. *Prog. NMR Spectr.*, **1994**, *26*, 27–58.

[499] Case, D.A. The use of chemical shifts and their anisotropies in biomolecular structure determination. *Curr. Opin. Struct. Biol.*, **1998**, *8*, 624–630.

[500] Torda, A.E.; Scheek, R.M.; VanGunsteren, W.F. Time-dependent distance restraints in molecular dynamics simulations. *Chem. Phys. Lett.*, **1989**, *157*, 289–294.

[501] Pearlman, D.A.; Kollman, P.A. Are time-averaged restraints necessary for nuclear magnetic resonance refinement? A model study for DNA. *J. Mol. Biol.*, **1991**, *220*, 457–479.

[502] Torda, A.E.; Brunne, R.M.; Huber, T.; Kessler, H.; van Gunsteren, W.F. Structure refinement using time-averaged J-coupling constant restraints. *J. Biomol. NMR*, **1993**, *3*, 55–66.

[503] Pearlman, D.A. How well to time-averaged J-coupling restraints work? *J. Biomol. NMR*, **1994**, *4*, 279–299.

[504] Pearlman, D.A. How is an NMR structure best defined? An analysis of molecular dynamics distance-based approaches. *J. Biomol. NMR*, **1994**, *4*, 1–16.

[505] Brünger, A.T.; Adams, P.D.; Clore, G.M.; Delano, W.L.; Gros, P.; Grosse-Kunstleve, R.W.; Jiang, J.-S.; Kuszewski, J.; Nilges, M.; Pannu, N.S.; Read, R.J.; Rice, L.M.; Simonson, T.; Warren, G.L. Crystallography and NMR system (CNS): A new software system for macromolecular structure determination. *Acta Cryst. D*, **1998**, *54*, 905–921.

[506] Yu, N.; Yennawar, H.P.; Merz, K.M. Jr. Refinement of protein crystal structures using energy restraints derived from linear-scaling quantum mechanics . *Acta Cryst. D*, **2005**, *61*, 322–332.

[507] Yu, N.; Li, X.; Cui, G.; Hayik, S.; Merz, K.M. Jr. Critical assessment of quantum mechanics based energy restraints in protein crystal structure refinement. *Prot. Sci.*, **2006**, *15*, 2773–2784.

[508] Yang, W.; Lee, T.-S. A density-matrix divide-and-conquer approach for electronic structure calculations of large molecules. *J. Chem. Phys.*, **1995**, *103*, 5674–5678.

[509] Wu, X.; Milne, J. L.; Borgnia, M. J.; Rostapshov, A. V.; Subramaniam, S.; Brooks, B. R. A core-weighted fitting method for docking atomic structures into low-resolution maps: application to cryo-electron microscopy. *J Struct Biol*, **2003**, *141*, 63–76.

[510] Milne, J. L.; Wu, X.; Borgnia, M. J.; Lengyel, J. S.; Brooks, B. R.; Shi, D.; Perham, R. N.; Subramaniam, S. Molecular structure of a 9-MDa icosahedral pyruvate dehydrogenase subcomplex containing the E2 and E3 enzymes using cryoelectron microscopy. *J Biol Chem*, **2006**, *281*, 4364–70.

[511] Wu, X.; Brooks, B. R. Modeling of Macromolecular assemblies with map objects. *Proc. 2007 Int. Conf. Bioinform. Comput. Biol.*, **2007**, *II*, 411–417.

[512] Khursigara, C. M.; Wu, X.; Zhang, P.; Lefman, J.; Subramaniam, S. Role of HAMP domains in chemotaxis signaling by bacterial chemoreceptors. *PNAS*, **2008**, *105*, 16555–60.

[513] Lengyel, J. S.; Stott, K. M.; Wu, X.; Brooks, A. B. R.and Balbo; Schuck, P.; Perham, R. N.; Subramaniam, S.; Milne, J. L. Extended polypeptide linkers establish the spatial architecture of a pyruvate dehydrogenase multienzyme complex. *Structure*, **2008**, *16*, 93–103.

[514] Khursigara, C. M.; Wu, X.; ; Subramaniam, S. Chemoreceptors in Caulobacter crescentus: trimers of receptor dimers in a partially ordered hexagonally packed array. *J Bacteriol*, **2008**, *190*, 6805–10.

[515] Elegheert, J.; Desfosses, A.; Shkumatov, A. V.; Wu, X.; Bracke, N.; Verstraete, K.; Van Craenenbroeck, K.; Brooks, B. R.; Svergun, D. I.; Vergauwen, B.; Gutsche, I.; Savvides, S. N. Extracellular complexes of the hematopoietic human and mouse CSF-1 receptor are driven by common assembly principles . *Structure*, **2011**, *19*, 1762–72.

[516] Khursigara, C. M.; Lan, G.; Neumann, S.; Wu, X.; Ravindran, S.; Borgnia, M. J.; Sourjik, V.; Milne, J.; Tu, Y.; Subramaniam, S. Lateral density of receptor arrays in the membrane plane influences sensitivity of the E. coli chemotaxis response. *Embo J*, **2011**, *30*, 1719–29.

[517] Wu, X.; Brooks, B. R. in *Microscopy: advances in scientific research and education*, Mendez-Vilas, A., Ed., pp 39–47. Formatex Research Center, Spain, 2014.

[518] Wu, X.; Brooks, B. R. in *Modern electron microscopy in physical and life science*, Janecek, Milos, Ed., chapter 12, pp 243–262. InTech, 2016.

[519] Bartesaghi, A.; Merk, A.; Banerjee, S.; Matthies, D.; Wu, X.; Milne, J.L.S.; Subramaniam, S. 2.2 a resolution cryo-em structure of beta-galactosidase in complex with a cell-permeant inhibitor. *Science*, **2015**, *348*, 1147–1151.

[520] Miranker, A.; Karplus, M. Functionality maps of binding sites: A multiple copy simultaneous search method. *Proteins: Str. Funct. Gen.*, **1991**, *11*, 29–34.

[521] Cheng, X.; Hornak, V.; Simmerling, C. Improved conformational sampling through an efficient combination of mean-field simulation approaches. *J. Phys. Chem. B*, **2004**, *108*.

[522] Simmerling, C.; Fox, T.; Kollman, P.A. Use of Locally Enhanced Sampling in Free Energy Calculations: Testing and Application of the alpha to beta Anomerization of Glucose. *J. Am. Chem. Soc.*, **1998**, *120*, 5771–5782.

[523] Straub, J.E.; Karplus, M. Enery partitioning in the classical time-dependent Hartree approximation. *J. Chem. Phys.*, **1991**, *94*, 6737.

[524] Ulitsky, A.; Elber, R. The thermal equilibrium aspects of the time-dependent Hartree and the locally enhanced sampling approximations: Formal properties, a correction, and computational examples for rare gas clusters. *J. Chem. Phys.*, **1993**, *98*, 3380.

[525] Schulman, L.S. *Techniques and Applications of Path Integration*. Wiley & Sons, New York, 1996.

[526] Chandler, D.; Wolynes, P.G. Exploiting the isomorphism between quantum theory and classical statistical mechanics of polyatomic fluids. *J. Chem. Phys.*, **1981**, *74*, 4078–4095.

[527] Ceperley, D.M. Path integrals in the theory of condensed helium. *Rev. Mod. Phys.*, **1995**, *67*, 279–355.

[528] Martyna, G.J.; Klein, M.L.; Tuckerman, M. Nosié-Hoover chains: The canonical ensemble via continuous dynamics. *J. Chem. Phys.*, **1992**, *97*, 2635.

[529] Berne, B.J.; Thirumalai, D. On the simulation of quantum systems: path integral methods. *Annu. Rev. Phys. Chem.*, **1986**, *37*, 401.

[530] Cao, J.; Berne, B.J. On energy estimators in path integral Monte Carlo simulations: Dependence of accuracy on algorithm. *J. Chem. Phys.*, **1989**, *91*, 6359–6366.

[531] Martyna, G.J.; Hughes, A.; Tuckerman, M.E. Molecular dynamics algorithms for path integrals at constant pressure. *J. Chem. Phys.*, **1999**, *110*, 3275.

[532] Voth, G.A. Path-integral centroid methods in quantum statistcal mechanics and dynamics. *Adv. Chem. Phys.*, **1996**, *93*, 135.

[533] Craig, I.R.; Manolopoulos, D.E. Quantum statistics and classical mechnanics: Real time correlation functions from ring polymer molecular dynamics. *J. Chem. Phys.*, **2004**, *121*, 3368.

[534] Cao, J.; Voth, G.A. The formulation of quantum statistical mechanics based on the Feynman path centroid density. IV. Algorithms for centroid molecular dynamics. *J. Chem. Phys.*, **1994**, *101*, 6168.

[535] Miller, T.F.; Manolopoulos, D.E. Quantum diffusion in liquid water from ring polymer molecular dynamics. *J. Chem. Phys.*, **2005**, *123*, 154504.

[536] Berne, B.J.; D. Harp, G. *Adv. Chem. Phys.*, **1970**, *17*, 63.

[537] Miller, W.H.; Schwartz, S.D.; Tromp, J.W. Quantum mechanical rate constants for bimolecular reactions. *J. Chem. Phys.*, **1983**, *79*, 4889–4898.

[538] Kubo, R.; Toda, M.; Hashitsume, N. *Statistical Physics II: Nonequilibrium Statistical Mechanics, 2nd ed.* Springer-Verlag, Heidelberg, 1991.

[539] Miller, W.H. *Adv. Chem. Phys.*, **1974**, *25*, 69.

[540] Miller, W.H. Including quantum effects in the dynamics of complex (i.e., large) molecular systems. *J. Chem. Phys.*, **2006**, *125*, 132305.

[541] Wang, H.; Sun, X.; Miller, W.H. Semiclassical approximations for the calculation of thermal rate constants for chemical reactions in complex molecular systems. *J. Chem. Phys.*, **1998**, *108*, 9726.

[542] Sun, X.; Wang, H.; H. Miller, W. Semiclassical theory of electronically nonadiabatic dynamics: Results of a linearized approximation to the initial value representation. *J. Chem. Phys.*, **1998**, *109*, 7064.

[543] Liu, J.; H. Miller, W. A simple model for the treatment of imaginary frequencies in chemical reaction rates and molecular liquids. *J. Chem. Phys.*, **2009**, *131*, 074113.

[544] Liu, J.; H. Miller, W.; Paesani, F.; Zhang, W.; A. Case, D. Quantum dynamical effects in liquid water: A semiclassical study on the diffusion and the infrared absorption spectrum. *J. Chem. Phys.*, **2009**, *131*, 164509.

[545] Liu, J.; Miller, W.H. Real time correlation function in a single phase space integral beyond the linearized semiclassical initial value representation. *J. Chem. Phys.*, **2007**, *126*, 234110.

[546] Liu, J.; Miller, W.H. Test of the consistency of various linearized semiclassical initial value time correlation functions in application to inelastic neutron scattering from liquid para-hydrogen. *J. Chem. Phys.*, **2008**, *128*, 144511.

[547] Shi, Q.; Giva, E. *J. Chem. Phys. A*, **2003**, *107*, 9059.

[548] Voth, G.A.; Chandler, D.; Miller, W.H. Rigorous Formulation of Quantum Transition State Theory and Its Dynamical Corrections. *J. Chem. Phys.*, **1989**, *91*, 7749–7760.

[549] Miller, W.H. Semiclassical limit of quantum mechanical transition state theory for nonseparable systems. *J. Chem. Phys.*, **1975**, *62*, 1899.

[550] Miller, W.H.; Zhao, Y.; Ceotto, M.; Yang, S. Quantum instanton approximation for thermal rate constants of chemical. *J. Chem. Phys.*, **2003**, *119*, 1329–1342.

[551] Yamamoto, T.; Miller, W.H. On the efficient path integral evaluation of thermal rate constants with the quantum instanton approximation. *J. Chem. Phys.*, **2004**, *120*, 3086–3099.

[552] Vaníček, J.; Miller, W.H.; Castillo, J.F.; Aoiz, F.J. Quantum-instanton evalution of the kinetic isotope effects. *J. Chem. Phys.*, **2005**, *123*, 054108.

[553] Vaníček, J.; Miller, W.H. Efficient estimators for quantum instanton evaluation of the kinetic isotope effects: application to the intramolecular hydrogen transfer in pentadiene. *J. Chem. Phys.*, **2007**, *127*, 114309.

[554] Yamamoto, T.; Miller, W.H. Path integral evaluation of the quantum instanton rate constant for proton transfer in a polar solvent. *J. Chem. Phys.*, **2005**, *122*, 044106.

[555] Cerutti, D.S.; Case, D.A. Multi-Level Ewald: A Hybrid Multigrid/Fast Fourier Transform Approach to the Electrostatic Particle-Mesh Problem. *J. Chem. Theory Comput.*, **2010**, *6*, 443–458.

[556] Roe, Daniel R.; Cheatham, T.E. III. PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data. *J. Chem. Theory Comput.*, **2013**, *9*, 3084–3095.

[557] Chatterjee, S.; Debenedetti, P. G.; Stillinger, F. H.; Lynden-Bell, R. M. A Computational Investigation of Thermodynamics, Structure, Dynamics and Solvation Behavior in Modified Water Models. *J. Chem. Phys.*, **2008**, *128*, 124511.

[558] Lazaridis, T. Inhomogeneous Fluid Approach to Solvation Thermodynamics. 1 Theory. *J. Phys. Chem. B*, **1998**, *102*, 3531–3541.

[559] Nguyen, C. N.; Kurtzman Young, T.; Gilson, M. K. Grid Inhomogeneous Solvation Theory: Hydration Structure and Thermodynamics of the Miniature Receptor cucurbit[7]uril. *J. Chem. Phys.*, **2012**, *137*, 044101.

[560] Humphrey, W.; Dalke, A.; Schulten, K. VMD Visual Molecular Dynamics. *J. Molec. Graph.*, **1996**, *14*, 33–38.

[561] Sindhikara, D. J.; Yoshida, N.; Hirata, F. Placevent: An Algorithm for Prediction of Explicit Solvent Atom Distribution-Application to HIV-1 Protease and F-ATP Synthase. *J. Comput. Chem.*, **2012**, *33*, 1536–1543.

[562] Chou, J.J.; Case, D.A.; Bax, A. Insights into the mobility of methyl-bearing side chains in proteins. *J. Am. Chem. Soc.*, **2003**, *125*, 8959–8966.

[563] Perez, C.; Lohr, F.; Ruterjans, H.; Schmidt, J.M. Self-Consistent Karplus Parameterization of (3)J couplings depending on the polypeptide side-chain torsion chi(1). *J. Am. Chem. Soc.*, **2001**, *123*, 7081–7093.

[564] Hunenberger, P.H.; Mark, A.E.; van Gunsteren, W.F. Fluctuation and Cross-correlation Analysis of Protein Motions Observed in Nanosecond Molecular Dynamics Simulations. *J. Mol. Biol.*, **1995**, *252*, 492–503.

[565] Prompers, J.J.; Brüschweiler, R. Dynamic and structural analysis of isotropically distributed molecular ensembles. *Proteins*, **2002**, *46*, 177–189.

[566] Prompers, J.J.; Brüschweiler, R. General framework for studying the dynamics of folded and nonfolded proteins by NMR relaxation spectroscopy and MD simulation. *J. Am. Chem. Soc.*, **2002**, *124*, 4522–4534.

[567] Connolly, M.L. Analytical molecular surface calculation. *J. Appl. Cryst.*, **1983**, *16*, 548–558.

[568] Hassan, M. A. El; Calladine, C. R. Two distinct modes of protein-induced bending in dna. *J. Mol. Biol.*, **1998**, *282*, 331–343.

[569] Lu, XJ; Olson, WK. 3dna: a software package for the analysis, rebuilding and visualization of three-dimensional nucleic acid structures. *NUCLEIC ACIDS RESEARCH*, **2003**, *31*, 5108–5121.

[570] Babcock, M.S.; Pednault, E.P.D.; Olson, W.K. Nucleic Acid Structure Analysis. *J. Mol. Biol.*, **1994**, *237*, 125–156.

[571] Olson, Wilma K.; Bansal, Manju; Burley, Stephen K.; Dickerson, Richard E.; Gerstein, Mark; Harvey, Stephen C.; Heinemann, Udo; Lu, Xiang-Jun; Neidle, Stephen; Shakked, Zippora; Sklenar, Heinz; Suzuki, Masashi; Tung, Chang-Shung; Westhof, Eric; Wolberger, Cynthia; Berman, Helen M. A standard reference frame for the description of nucleic acid base-pair geometry. *J. Mol. Biol.*, **2001**, *313*, 229–237.

[572] Altona, C; Sundaralingam, M. Conformational analysis of the sugar ring in nucleosides and nucleotides. a new description using the concept of pseudorotation. *J Am Chem Soc*, **1972**, *94*, 8205–8212.

[573] Harvey, SC; Prabhakaran, M. Ribose puckering - structure, dynamics, energetics, and the pseudorotation cycle. *J Am Chem Soc*, **1986**, *108*, 6128–6136.

[574] Cremer, D; Pople, JA. A general definition of ring puckering coordinates. *J Am Chem Soc*, **1975**, *97*, 1354–1358.

[575] Kabsch, W.; Sander, C. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, **1983**, *22*, 2577–2637.

[576] Cui, G.; Swails, J. M.; Manas, E. S. SPAM: A Simple Approach for Profiling Bound Water Molecules. *J. Chem. Theory Comput.*, **2013**, *9*, 5539–5549.

[577] Ester, M.; Kriegel, H.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proc. Second Int. Conf. Knowledge Disc. Data Mining (KDD-96)*, **1996**, pp 226–231.

[578] Rodriguez, Alex; Laio, Alessandro. Clustering by fast search and find of density peaks. *Science*, **2014**, *344*, 1492–1496.

[579] Bakan, A.; Meireles, L.M.; Bahar, I. ProDy: Protein Dynamics Inferred from Theory and Experiments. *Bioinformatics*, **2011**, *27*, 1575–1577.

[580] Roe, Daniel R.; Bergonzo, Christina; III, Thomas E. Cheatham. Evaluation of enhanced sampling provided by accelerated molecular dynamics with hamiltonian replica exchange methods. *J. Phys. Chem. B*, **2014**, *118*, 3543–3552.

[581] Galindo-Murillo, Rodrigo; Roe, Daniel R.; Cheatham, T.E. III. Convergence and reproducibility in molecular dynamics simulations of the DNA duplex d(GCACGAACGAACGAACGC). *Biochim. Biophys. Acta*, **2015**, *1850*, 1041–1058.

[582] Wong, V.; Case, D.A. Evaluating rotational diffusion from protein md simulations. *J. Phys. Chem. B*, **2008**, *112*, 6013–6024.

[583] Schneider, B.; Neidle, S.; Berman, H. M. Conformations of the sugar-phosphate backbone in helical dna crystal structures. *Biopolymers*, **1997**, *42*, 113–124.

[584] Schneider, B.; Moravek, Z.; Berman, H. M. Rna conformational classes. *Nucleic Acids Res.*, **2004**, *32*, 1666–1677.

[585] Torrence, C.; Compo, G. P. A practical guide to wavelet analysis. *Bull. Am. Meteorol. Soc.*, **1998**, *79*, 61–78.

[586] Benson, N. C.; Dagget, V. Wavelet analysis of protein motion. *Int. J. Wavelets Multi.*, **2012**, *10*.

[587] Heidari, Z.; Roe, D. R.; Galindo-Murillo, R.; Ghasemi, J. B.; Cheatham, T. E. III. Using Wavelet Analysis To Assist in Identification of Significant Events in Molecular Dynamics Simulations. *J. Chem. Inf. Model.*, **2016**, *56*, 1282–1291.

[588] Nguyen, Hai; Roe, Daniel R.; Swails, Jason M.; Case., David A. PYTRAJ: Interactive data analysis for molecular dynamics simulations. *Manuscript in preparation*, **2016**.

[589] Miller, Bill R.; McGee, T. Dwight; Swails, Jason M.; Homeyer, Nadine; Gohlke, Holger; Roitberg, Adrian E. MMPBSA.py: An Efficient Program for End-State Free Energy Calculations. *J. Chem. Theory Comput.*, **2012**, *8*, 3314–3321.

[590] Srinivasan, J.; Cheatham, T.E. III; Cieplak, P.; Kollman, P.; Case, D.A. Continuum solvent studies of the stability of DNA, RNA, and phosphoramidate–DNA helices. *J. Am. Chem. Soc.*, **1998**, *120*, 9401–9409.

[591] Kollman, P.A.; Massova, I.; Reyes, C.; Kuhn, B.; Huo, S.; Chong, L.; Lee, M.; Lee, T.; Duan, Y.; Wang, W.; Donini, O.; Cieplak, P.; Srinivasan, J.; Case, D.A.; Cheatham, T.E. III. Calculating structures and free energies of complex molecules: Combining molecular mechanics and continuum models. *Accts. Chem. Res.*, **2000**, *33*, 889–897.

[592] Homeyer, N.; Gohlke, H. Free energy calculations by the molecular mechanics poisson-boltzmann surface area method. *Mol. Informatics*, **2012**, *DOI: 10.1002/minf.201100135*.

[593] Wang, W.; Kollman, P. Free energy calculations on dimer stability of the HIV protease using molecular dynamics and a continuum solvent model. *J. Mol. Biol.*, **2000**, *303*, 567.

[594] Reyes, C.; Kollman, P. Structure and thermodynamics of RNA-protein binding: Using molecular dynamics and free energy analyses to calculate the free energies of binding and conformational change. *J. Mol. Biol.*, **2000**, *297*, 1145–1158.

[595] Lee, M.R.; Duan, Y.; Kollman, P.A. Use of MM-PB/SA in estimating the free energies of proteins: Application to native, intermediates, and unfolded vilin headpiece. *Proteins*, **2000**, *39*, 309–316.

[596] Wang, J.; Morin, P.; Wang, W.; Kollman, P.A. Use of MM-PBSA in reproducing the binding free energies to HIV-1 RT of TIBO derivatives and predicting the binding mode to HIV-1 RT of efavirenz by docking and MM-PBSA. *J. Am. Chem. Soc.*, **2001**, *123*, 5221–5230.

[597] Marinelli, L.; Cosconati, S.; Steinbrecher, T.; Limongelli, V.; Bertamino, A.; Novellino, E.; Case, D.A. Homology Modeling of NR2B Modulatory Domain of NMDA Receptor and Analysis of Ifenprodil Binding. *ChemMedChem*, **2007**, *2,*, 1498–1510.

[598] The open babel package, version 1.6. **2004**, *http://openbabel.sourceforge.net*.

[599] Homeyer, N.; Gohlke, H. FEW - A workflow tool for free energy calculations of ligand binding. *J. Comput. Chem.*, **2013**, *34*, 965–973.

[600] Jo, S.; Kim, T.; Im, W. Automated builder and database of protein/membrane complexes for molecular dynamics simulations. *PLoS One*, **2007**, *2*, e880.

[601] Jo, S.; Kim, T.; Iyer, V.G.; W., Im. CHARMM-GUI: a web-based graphical user interface for CHARMM. *J. Comput. Chem.*, **2008**, *29*, 1859–1865.

[602] Jo, S.; Lim, J.B.; Klauda, J.B.; Im, W. CHARMM-GUI Membrane Builder for mixed bilayers and its application to yeast membranes. *Biophys. J.*, **2009**, *97*, 50–58.

[603] Wu, E.L.; Cheng, X.; Jo, S.; Rui, H.; Song, K.C.; Davila-Contreras, E.M.; Qi, Y.; Lee, J.; Monje-Galvan, V.; Venable, R.M.; Klauda, J.B.; W., Im. CHARMM-GUI Membrane Builder toward realistic biological membrane simulations. *J. Comput. Chem.*, **2014**, *35*, 1997–2004.

[604] Domanski, J.; Stansfeld, P.; Sansom, M.S.P.; Beckstein, O. Lipidbook: A Public Repository for Force Field Parameters Used in Membrane Simulations. *J. Membrane Biol.*, **2010**, *236*, 255–258.

[605] Izadi, Saeed; Aguilar, Boris; Onufriev, Alexey V. Protein-ligand electrostatic binding free energies from explicit and implicit solvation. *J. Chem. Theory Comput.*, **2015**, *11*, 4450–4459.

[606] Metz, A. *Goethe University (Frankfurt am Main)*, **2006**.

[607] Baker, N.A.; Sept, D.; Simpson, J.; Holst, M.J.; J.A., McCammon. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc. Natl. Acad. Sci. U.S.A.*, **2001**, *98*, 10037–10041.

[608] Holst, M.; Saied, F. Multigrid solution of the Poisson-Boltzmann equation. *J. Comput. Chem.*, **1993**, *14*, 105–113.

[609] Holst, M.; Saied, F. Numerical solution of the nonlinear Poisson-Boltzmann equation: Developing more robust and efficient methods. *J. Comput. Chem.*, **1995**, *16*, 337–364.

[610] Holst, M. Adaptive numerical treatment of elliptic systems on manifolds. *Adv. Comput. Math.*, **2001**, *15*, 139–191.

[611] Bank, R.; Holst, M. A New Paradigm for Parallel Adaptive Meshing Algorithms. *SIAM Review*, **2003**, *45*, 291–323.

[612] Callenberg, K.M.; Choudhary, O.P.; de Forest, G.L.; Gohara, D.W.; Baker, N.A.; Grabe, M. APBSmem: A graphical interface for electrostatic calculations at the membrane. *PLoS One*, **2010**, *5*, e12722.

[613] Grabe, M.; Lecar, H.; Jan, Y.N.; L.Y., Jan. A quantitative assessment of models for voltage-dependent gating of ion channels. *Proc. Natl. Acad. Sci. U.S.A.*, **2004**, *101*, 17640–17645.

[614] Homeyer, N.; Gohlke, H. Extension of the free energy workflow FEW towards implicit solvent/implicit membrane MM-PBSA calculations. *BBA - Gen. Subjects*, **2015**, *1850*, 972–982.

[615] Nymeyer, H.; Zhou, H.X. A method to determine dielectric constants in nonhomogeneous systems: application to biological membranes. *Biophys. J.*, **2008**, *94*, 1185–1193.

[616] Stern, H.A.; Feller, S.E. Calculation of the dielectric permittivity profile for a nonuniform system: application to a lipid bilayer simulation. *J. Chem. Phys.*, **2003**, *118*, 3401–3412.

[617] Åqvist, J.; Medina, C.; Samuelsson, J. E. A new method for predicting binding affinity in computer-aided drug design. *Protein Eng.*, **1994**, *7*, 385–391.

[618] Åqvist, J.; Luzhkov, V. B.; Brandsdal, B. O. Ligand binding affinities from MD simulations. *Acc. Chem. Res.*, **2002**, *35*, 358–365.

[619] van Lipzig, M. M.; ter Laak, A. M.; Jongejan, A.; Vermeulen, N. P.; Wamelink, M.; Geerke, D.; Meerman, J. H. Prediction of ligand binding affinity and orientation of xenoestrogens to the estrogen receptor by molecular dynamics simulations and the linear interaction energy method. *J. Med. Chem.*, **2004**, *47*, 1018–1030.

[620] Wallnoefer, H. G.; Liedl, K. R.; Fox, T. A challenging system: free energy prediction for factor Xa. *J. Comput. Chem.*, **2011**, *32*, 1743–1752.

[621] Brandsdal, B. O.; Österberg, F.; Almlöf, M.; Feierberg, I.; Luzhkov, V. B.; Åqvist, J. Free energy calculations and ligand binding. *Adv. Protein Chem.*, **2003**, *66*, 123–158.

[622] Wang, W.; Wang, J.; Kollman, P. A. What determines the van der Waals coefficient beta in the LIE (linear interaction energy) method to estimate binding free energies using molecular dynamics simulations? *Proteins: Struct., Funct., Genet.*, **1999**, *34*, 395–402.

[623] Jones-Hertzog, D. K.; Jorgensen, W. L. Binding affinities for sulfonamide inhibitors with human thrombin using Monte Carlo simulations with a linear response method. *J. Med. Chem.*, **1997**, *40*, 1539–1549.

[624] Lamb, M. L.; Tirado-Rives, J.; Jorgensen, W. L. Estimation of the binding affinities of FKBP12 inhibitors using a linear response method. *Bioorg. Med. Chem.*, **1999**, *7*, 851–860.

[625] Yang, W.; Bitetti-Putzer, R.; M., Karplus. Free energy simulations: Use of reverse cumulative averaging to determine the equilibrated region and the time required for convergence. *J. Chem. Phys.*, **2004**, *120*, 2618–2628.

[626] Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, **1956**, *7*, 48–50.

[627] *ROCS, OpenEye Scientific Software, Santa Fe*, *http://www.eyesopen.com*.

[628] Hawkins, P. C. D.; Skillman, A. G.; Nicholls, A. Comparison of shape-matching and docking as virtual screening tools. *J. Med. Chem.*, **2007**, *50*, 74–82.

[629] Janowski, P. A.; Cerutti, D.S.; Holton, J. M.; Case, D. A. Peptide crystal simulations reveal hidden dynamics. *J. Am. Chem. Soc.*, **2013**, *135*, 7938–7948.

[630] Park, S.; Bardhan, J.P.; Roux, B.; Makowski, L. Simulated X-ray scattering of protein solutions using explicit-solvent models. *J. Chem. Phys.*, **2009**, *130*, 134114.

[631] Nguyen, H.T.; Pabit, S.A.; Meisburger, S.P.; Pollack, L.; Case, D.A. Accurate small and wide angle X-ray scattering profiles from atomic models of proteins and nucleic acids. *J. Chem. Phys.*, **2014**, *141*, 22D508.

[632] Major, F.; Turcotte, M.; Gautheret, D.; Lapalme, G.; Fillon, E.; Cedergren, R. The Combination of Symbolic and Numerical Computation for Three-Dimensional Modeling of RNA. *Science*, **1991**, *253*, 1255–1260.

[633] Gautheret, D.; Major, F.; Cedergren, R. Modeling the three-dimensional structure of RNA using discrete nucleotide conformational sets. *J. Mol. Biol.*, **1993**, *229*, 1049–1064.

[634] Turcotte, M.; Lapalme, G.; Major, F. Exploring the conformations of nucleic acids. *J. Funct. Program.*, **1995**, *5*, 443–460.

[635] Erie, D.A.; Breslauer, K.J.; Olson, W.K. A Monte Carlo Method for Generating Structures of Short Single-Stranded DNA Sequences. *Biopolymers*, **1993**, *33*, 75–105.

[636] Tung, C.-S.; Carter, E.S. II. Nucleic acid modeling tool (NAMOT): an interactive graphic tool for modeling nucleic acid structures. *CABIOS*, **1994**, *10*, 427–433.

[637] Carter, E.S. II; Tung, C.-S. NAMOT2–a redesigned nucleic acid modeling tool: construction of non-canonical DNA structures. *CABIOS*, **1996**, *12*, 25–30.

[638] Zhurkin, V.B.; P. Lysov, Yu.; Ivanov, V.I. Different Families of Double Stranded Conformations of DNA as Revealed by Computer Calculations. *Biopolymers*, **1978**, *17*, 277–312.

[639] Lavery, R.; Zakrzewska, K.; Skelnar, H. JUMNA (junction minimisation of nucleic acids). *Comp. Phys. Commun.*, **1995**, *91*, 135–158.

[640] Gabarro-Arpa, J.; Cognet, J.A.H.; Le Bret, M. Object Command Language: a formalism to build molecule models and to analyze structural parameters in macromolecules, with applications to nucleic acids. *J. Mol. Graph.*, **1992**, *10*, 166–173.

[641] Le Bret, M.; Gabarro-Arpa, J.; Gilbert, J.C.; Lemarechal, C. MORCAD an object-oriented molecular modeling package. *J. Chim. Phys.*, **1991**, *88*, 2489–2496.

[642] Crippen, G.M.; Havel, T.F. *Distance Geometry and Molecular Conformation*. Research Studies Press, Taunton, England, 1988.

[643] Spellmeyer, D.C.; Wong, A.K.; Bower, M.J.; Blaney, J.M. Conformational analysis using distance geometry methods. *J. Mol. Graph. Model.*, **1997**, *15*, 18–36.

[644] Hodsdon, M.E.; Ponder, J.W.; Cistola, D.P. The NMR solution structure of intestinal fatty acid-binding protein complexed with palmitate: Application of a novel distance geometry algorithm. *J. Mol. Biol.*, **1996**, *264*, 585–602.

[645] Macke, T.; Chen, S.-M.; Chazin, W.J. in *Structure and Function, Volume 1: Nucleic Acids*, Sarma, R.H.; Sarma, M.H., Eds., pp 213–227. Adenine Press, Albany, 1992.

[646] Potts, B.C.M.; Smith, J.; Akke, M.; Macke, T.J.; Okazaki, K.; Hidaka, H.; Case, D.A.; Chazin, W.J. The structure of calcyclin reveals a novel homodimeric fold S100 Ca^{2+}-binding proteins. *Nature Struct. Biol.*, **1995**, *2*, 790–796.

[647] Love, J.J.; Li, X.; Case, D.A.; Giese, K.; Grosschedl, R.; Wright, P.E. DNA recognition and bending by the architectural transcription factor LEF-1: NMR structure of the HMG domain complexed with DNA. *Nature*, **1995**, *376*, 791–795.

[648] Gurbiel, R.J.; Doan, P.E.; Gassner, G.T.; Macke, T.J.; Case, D.A.; Ohnishi, T.; Fee, J.A.; Ballou, D.P.; Hoffman, B.M. Active site structure of Rieske-type proteins: Electron nuclear double resonance studies of isotopically labeled phthalate dioxygenase from *Pseudomonas cepacia and Rieske protein from Rhodobacter capsulatus* and molecular modeling studies of a Rieske center. *Biochemistry*, **1996**, *35*, 7834–7845.

[649] Macke, T.J. *NAB, a Language for Molecular Manipulation*. Ph.D. thesis, The Scripps Research Institute, 1996.

[650] Dickerson, R.E. Definitions and Nomenclature of Nucleic Acid Structure Parameters. *J. Biomol. Struct. Dyn.*, **1989**, *6*, 627–634.

[651] Tan, R.; Harvey, S. Molecular Mechanics Model of Supercoiled DNA. *J. Mol. Biol.*, **1989**, *205*, 573–591.

[652] Havel, T.F.; Kuntz, I.D.; Crippen, G.M. The theory and practice of distance geometry. *Bull. Math. Biol.*, **1983**, *45*, 665–720.

[653] Havel, T.F. An evaluation of computational strategies for use in the determination of protein structure from distance constraints obtained by nuclear magnetic resonance. *Prog. Biophys. Mol. Biol.*, **1991**, *56*, 43–78.

[654] Kuszewski, J.; Nilges, M.; Brünger, A.T. Sampling and efficiency of metric matrix distance geometry: A novel partial metrization algorithm. *J. Biomolec. NMR*, **1992**, *2*, 33–56.

[655] deGroot, B.L.; van Aalten, D.M.F.; Scheek, R.M.; Amadei, A.; Vriend, G.; Berendsen, H.J.C. Prediction of protein conformational freedom from distance constraints. *Proteins*, **1997**, *29*, 240–251.

[656] Agrafiotis, D.K. Stochastic Proximity Embedding. *J. Comput. Chem.*, **2003**, *24*, 1215–1221.

[657] Saenger, W. in *Principles of Nucleic Acid Structure*, p 120. Springer-Verlag, New York, 1984.

[658] Brooks, C.; Brünger, A.; Karplus, M. Active site dynamics in protein molecules: A stochastic boundary molecular-dynamics approach. *Biopolymers*, **1985**, *24*, 843–865.

[659] Nguyen, D.T.; Case, D.A. On finding stationary states on large-molecule potential energy surfaces. *J. Phys. Chem.*, **1985**, *89*, 4020–4026.

[660] Shi, Z.J.; Shen, J. New inexact line search method for unconstrained optimization. *J. Optim. Theory Appl.*, **2005**, *127*, 425–446.

[661] Izadi, Saeed; Anandakrishnan, Ramu; Onufriev, Alexey V. Implicit solvent model for Million-Atom atomistic simulations: Insights into the organization of 30-nm chromatin fiber. *J. Chem. Theory Comput.*, **2016**, *12*, 5946–5959.

[662] Anandakrishnan, Ramu; Onufriev, Alexey V. An N log N approximation based on the natural organization of biomolecules for speeding up the computation of long range interactions. *J. Comput. Chem.*, **2010**, *31*, 691–706.

[663] Anandakrishnan, R.; Daga, M.; Onufriev, A.V. An n log n Generalized Born Approximation. *J. Chem. Theory Comput.*, **2011**, *7*, 544–559.

[664] Wang, J.; Wang, W.; Kollman, P.A.; Case, D.A. Automatic atom type and bond type perception in molecular mechanical. *J. Mol. Graphics Model.*, **2006**, *25*, 247–260.

[665] Chong, Lillian T.; Duan, Y.; Wang, L.; Massova, I.; Kollman, P.A. Molecular dynamics and free-energy calculations applied to affinity maturation in antibody 48G7. *Proc. Natl. Acad. Sci. USA*, **1999**, *96*, 14330–14335.

[666] Huo, S.; Massova, I.; Kollman, P.A. Computational Alanine Scanning of the 1:1 Human Growth Hormone-Receptor Complex. *J. Comput. Chem.*, **2002**, *23*, 15–27.

[667] Hingerty, B.E.; Figueroa, Samuel; L. Hayden, Thomas; Broyde, Suse. Prediction of DNA Structure from Sequence: A Build-up Technique. *Biopolymers*, **1989**, *28*, 1195–1222.

[668] Pearlman, D.A.; Kim, Sung-Hou. Conformational Studies of Nucleic Acids I. A Rapid and Direct Method for Generating Coordinates from the Pseudorotation Angle. *J. Biomol. Struct. Dyn.*, **1985**, *3*, 85–98.

[669] Pearlman, D.A.; Kim, Sung-Hou. Conformational Studies of Nucleic Acids II. The Conformational Energetics of Commonly Occurring Nucleosides. *J. Biomol. Struct. Dyn.*, **1985**, *3*, 99–125.

[670] Pearlman, D.A.; Kim, S.-H. Conformational Studies of Nucleic Acids: III. Empirical Multiple Correlation Functions for Nucleic Acid Torsion Angles. *J. Biomol. Struct. Dyn.*, **1986**, *4*, 49–67.

[671] Pearlman, D.A.; Kim, S.-H. Conformational Studies of Nucleic Acids: IV. The Conformational Energetics of Oligonucleotides: d(ApApApA) and ApApApA. *J. Biomol. Struct. Dyn.*, **1986**, *4*, 69–98.

[672] Pearlman, D.A.; Kim, S.-H. Conformational Studies of Nueleic Acids. V. Sequence Specificities of in the Conformational Energetics of Oligonucleotides: The Homo-Tetramers. *Biopolymers*, **1988**, *27*, 59–77.

[673] Schlick, T. A Modular Strategy for Generating Starting Conformations and Data Structures of Polynucleotide Helices for Potential Energy Calculations. *J. Comput. Chem.*, **1988**, *9*, 861–889.

[674] Lavery, R. in *Unusual DNA Structures*, Wells, R. D.; Harvey, S. C., Eds. Springer-Verlag, New York, 1988.

[675] Shen, L.; Tinoco, I. The Structure of an RNA Pseudoknot that Causes Efficient Frameshift in Mouse Mammary Tumor Virus. *J. Mol. Biol.*, **1995**, *247*, 963–978.

[676] Hubbard, J.M.; Hearst, J.E. Predicting the Three-Dimensional Folding of Transfer RNA with a Computer Modeling Protocol. *Biochemsitry*, **1991**, *30*, 5458–5465.

[677] Chou, S.-H.; Zhu, L.; Reid, B.R. The Unusual Structure of the Human Centromere (GGA)2 Motif. *J. Mol. Biol.*, **1994**, *244*, 259–268.

[678] Levitt, M. Detailed Molecular Model for Transfer Ribonucleic Acid. *Nature*, **1969**, *224*, 759–763.

[679] Hubbard, J.M.; Hearst, J.E. Computer Modeling 16S Ribosomal RNA. *J. Mol. Biol.*, **1991**, *221*, 889–907.

[680] Schlick, T.; Olson, W.K. Supercoiled DNA Energetics and Dynamics by Computer Simulation. *J. Mol. Biol.*, **1992**, *223*, 1089–1119.

[681] Holbrook, S.R.; Sussman, J.L.; Warrant, R.W.; Kim, S.-H. Crystal Structure of Yeast Phenylalanine Transfer RNA II. Structural Features and and Functional Implications. *J. Mol. Biol.*, **1978**, *123*, 631–660.

[682] Conner, B.N.; Yoon, C.; Dickerson, J.L; Dickerson, R.E. Helix Geometry and Hydration in an A-DNA Tetramer: C-C-G-G. *J. Mol. Biol.*, **1984**, *174*, 663–695.

[683] Brünger, A.T. *X-PLOR: A System for Cystallography and NMR, Version 3.1.* Yale University, New Haven, CT, 1992.

[684] Wyatt, J.R.; Puglisi, J.D.; Tinoco Jr., I. RNA Pseudoknots. Stability and Loop Size Requirements. *J. Mol. Biol.*, **1990**, *214*, 455–470.

[685] Puglisi, J.D.; Wyatt, J.R.; Tinoco Jr., I. Conformation of an RNA Pseudoknot. *J. Mol. Biol.*, **1990**, *214*, 437–453.

[686] Kuila, G.; Fee, J.A.; Schoonover, J.R.; Woodruff, W.H. Resonance Raman Spectra of the [2Fe-2S] Clusters of the Rieske Protein from Thermus and Phthalate Dioxygenase from Pseudomonas. *J. Am. Chem. Soc.*, **1987**, *109*, 1559–1561.

[687] Lewin, B. in *Genes IV*, pp 409–425. Cell Press, Cambridge, Mass., 1990.

[688] Press, W.H.; Teukolsky, S.A.; Vettering, W.T.; Flannery, B.P. in *Numerical Recipes in C*, pp 113–117. Cambridge, New York, 1992.

[689] Zhurkin, V.B.; Raghunathan, G.; Ulynaov, N.B.; Camerini-Otero, R.D.; Jernigan, R.L. A Parallel DNA Triplex as a Model for the Intermediate in Homologous Recombination. *J. Mol. Biol.*, **1994**, *239*, 181–200.

[690] van Gunsteren, W.F.; Weiner, P.K.; Wilkinson, A.J. eds. *Computer Simulations of Biomolecular Systems, Vol. 2.* ESCOM Science Publishers, Leiden, 1993.

[691] Åqvist, J.; Warshel, A. Computer simulation of the initial proton-transfer step in human carbonic anhydrase-I. *J. Mol. Biol.*, **1992**, *224*, 7–14.

[692] Berendsen, H.J.C.; Postma, J.P.M.; van Gunsteren, W.F.; DiNola, A.; Haak, J.R. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.*, **1984**, *81*, 3684–3690.

[693] Wishart, D.S.; Case, D.A. Use of chemical shifts in macromolecular structure determination. *Meth. Enzymol.*, **2001**, *338*, 3–34.

[694] Elber, R.; Karplus, M. Enhanced sampling in molecular dynamics. Use of the time-dependent Hartree approximation for a simulation of carbon monoxide diffusion through myoglobin. *J. Am. Chem. Soc.*, **1990**, *112*, 9161–9175.

[695] Roitberg, A.; Elber, R. Modeling side chains in peptides and proteins: Application of the locally enhanced sampling and the simulated annealing methods to find minimum energy conformations. *J. Chem. Phys.*, **1991**, *95*, 9277.

[696] Simmerling, C.; Miller, J.L.; Kollman, P.A. Combined locally enhanced sampling and particle mesh Ewald as a strategy to locate the experimental structure of a nonhelical nucleic acid. *J. Am. Chem. Soc.*, **1998**, *120*, 7149–7155.

[697] Simmerling, C.; Lee, M.R.; Ortiz, A.R.; Kolinski, A.; Skolnick, J.; Kollman, P.A. Combining MONSSTER and LES/PME to Predict Protein Structure from Amino Acid Sequence: Application to the Small Protein CMTI-1. *J. Am. Chem. Soc.*, **2000**, *122*, 8392–8402.

[698] Simmerling, C.; Elber, R. Hydrophobic "collapse" in a cyclic hexapeptide: Computer simulations of CHDLFC and CAAAAC in water. *J. Am. Chem. Soc.*, **1994**, *116*, 2534–2547.

[699] Ross, W.S.; Hardin, C.C. Ion-induced stabilization of the G-DNA quadruplex: Free energy perturbation studies. *J. Am. Chem. Soc.*, **1994**, *116*, 6070–6080.

[700] Vedani, A.; Huhta, D.W. A new force field for modeling metalloproteins. *J. Am. Chem. Soc.*, **1990**, *112*, 4759–4767.

[701] Veenstra, D.L.; Ferguson, D.M.; Kollman, P.A. How transferable are hydrogen parameters in molecular mechanics calculations? *J. Comput. Chem.*, **1992**, *13*, 971–978.

[702] Allen, F.H.; Kennard, O.; Watson, D.G.; Brammer, L.; Orpen, A.G.; Taylor, R. *J. Chem. Soc. Perkin Trans. II*, **1987**, pp S1–S19.

[703] Harmony, M.D.; Laurie, R.W.; Kuczkowski, R.L.; Schwendemann, R.H.; Ramsay, D.A.; Lovas, F.J.; Lafferty, W.J.; Maki, A.G. *J. Phys. Chem. Ref. Data*, **1979**, *8*, 619.

[704] Hopfinger, A.J.; Pearlstein, R.A. Molecular mechanics force-field parameterization procedures. *J. Comput. Chem.*, **1985**, *5*, 486–499.

[705] Cannon, J.F. AMBER force-field parameters for guanosine triphosphate and its imido and methylene analogs. *J. Comput. Chem.*, **1993**, *14*, 995–1005.

[706] Cornell, W.D.; Cieplak, P.; Bayly, C.I.; Kollman, P.A. Application of RESP charges to calculate conformational energies, hydrogen bond energies and free energies of solvation. *J. Am. Chem. Soc.*, **1993**, *115*, 9620–9631.

[707] Howard, A.E.; Cieplak, P.; Kollman, P.A. A molecular mechanical model that reproduces the relative energies for chair and twist-boat conformations of 1,3-dioxanes. *J. Comp. Chem.*, **1995**, *16*, 243–261.

[708] St.-Amant, A.; Cornell, W.D.; Kollman, P.A.; Halgren, T.A. Calculation of molecular geometries, relative conformational energies, dipole moments, and molecular electrostatic potential fitted charges of small organic molecules of biochemical interest by density functional theory. *J. Comput. Chem.*, **1995**, *16*, 1483–1506.

[709] Halgren, T.A. Merck Molecular Force Field (MMFF94). Part I-V. *J. Comput. Chem.*, **1996**, *17*, 490–641.

[710] Beveridge, D.L.; DiCapua, F.M. Free energy simulation via molecular simulations: Applications to chemical and biomolecular systems. *Annu. Rev. Biophys. Biophys. Chem.*, **1989**, *18*, 431–492.

[711] Chipot, C.; Kollman, P.A.; Pearlman, D.A. Alternative approaches to potential of mean force calculations: free energy perturbation versus thermodynamics integration. Case study of some representative nonpolar interactions. *J. Comput. Chem*, **1996**, *17*, 1112–1131.

[712] Pearlman, D.A.; Kollman, P.A. The overlooked bond-stretching contribution in free energy perturbation calculations. *J. Chem. Phys.*, **1991**, *94*, 4532–4545.

[713] Pearlman, D.A. Determining the contributions of constraints in free energy calculations: Development, characterization, and recommendations. *J. Chem. Phys.*, **1993**, *98*, 8946–8957.

[714] Pearlman, D.A. Free energy derivatives: A new method for probing the convergence problem in free energy calculations. *J. Comput. Chem.*, **1994**, *15*, 105–123.

[715] Pearlman, D.A. A comparison of alternative approaches to free energy calculations. *J. Phys. Chem.*, **1994**, *98*, 1487–1493.

[716] Pearlman, D.A.; Rao, B.G. in *Encyclopedia of Computational Chemistry*, von R. Schleyer, P.; Allinger, N.L.; Clark, T.; Gasteiger, J.; Kollman, P.A.; H.F. Schaefer, III, Eds., pp 1036–1061. John Wiley, Chichester, 1998.

[717] Radmer, R.J.; Kollman, P.A. Free energy calculation methods: A theoretical and empirical comparison of numerical errors and a new method for qualitative estimates of free energy changes. *J. Comput. Chem.*, **1997**, *18*, 902–919.

[718] Pearlman, D.A.; Kollman, P.A. A new method for carrying out free energy perturbation calculations: dynamically modified windows. *J. Chem. PHys.*, **1989**, *90*, 2460–2470.

[719] Yu, H.-A.; Karplus, M. A thermodynamic analysis of solvation. *J. Chem. Phys.*, **1988**, *89*, 2366–2379.

[720] Hummer, G. Fast-growth thermodynamic integration: Error and efficiency analysis. *J. Chem. Phys.*, **2001**, *114*, 7330–7337.

[721] Fleischman, S.H.; Brooks, C.L. III. Thermodynamic calculations on biological systems: Solution properties of alcohols and alkanes. *J. Chem. Phys.*, **1988**, *87*, 221–234.

[722] Vincent, J.J.; Merz, K.M. Jr. A highly portable parallel implementation of AMBER4 using the message passing interface standard. *J. Comput. Chem.*, **1995**, *16*, 1420–1427.

[723] Radmer, R.; Kollman, P. The application of three approximate free energy calculations methods to structure based ligand design: Trypsin and its complex with inhibitors. *J. Comput.-Aided Mol. Design*, **1998**, *12*, 215–228.

[724] Niketic, S.R.; Rasmussen, K. *The Consistent Force Field: A Documentation*. Springer-Verlag, New York, 1977.

[725] Cerjan, C.; Miller, W.H. On finding transition states. *J. Chem. Phys.*, **1981**, *75*, 2800.

[726] Nguyen, D.T.; Case, D.A. On finding stationary states on large-molecule potential energy surfaces. *J. Phys. Chem.*, **1985**, *89*, 4020–4026.

[727] Lamm, G.; Szabo, A. Langevin modes of macromolecules. *J. Chem. Phys.*, **1986**, *85*, 7334–7348.

[728] Kottalam, J.; Case, D.A. Langevin modes of macromolecules: application to crambin and DNA hexamers. *Biopolymers*, **1990**, *29*, 1409–1421.

[729] Darden, T.; Pearlman, D.; Pedersen, L.G. Ionic charging free energies: Spherical versus periodic boundary conditions. *J. Chem. Phys.*, **1998**, *109*, 10921–10935.

[730] Levy, R.M.; Karplus, M.; Kushick, J.; Perahia, D. Evaluation of the configurational entropy for proteins: Application to molecular dynamics simulations of an α-helix. *Macromolecules*, **1984**, *17*, 1370–1374.

[731] Arnott, S.; Campbell-Smith, P.J.; Chandrasekaran, R. in *Handbook of Biochemistry and Molecular Biology, 3rd ed. Nucleic*, Fasman, G.P., Ed., pp 411–422. CRC Press, Cleveland, 1976.

[732] Cheatham, T.E. III; Brooks, B.R.; Kollman, P.A. in *Current Protocols in Nucleic Acid Chemistry*, pp Sections 7.5, 7.8, 7.9, 7.10. Wiley, New York, 1999.

[733] Press, W.H.; Flannery, B.P.; Teukolsky, S.A.; Vetterling, W.T. in *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, 1989.

[734] Beroza, P.; Case, D.A. Calculations of proton-binding thermodynamics in proteins. *Meth. Enzymol.*, **1998**, *295*, 170–189.

[735] Madura, J.D.; Davis, M.E.; Gilson, M.K.; Wade, R.C.; Luty, B.A.; McCammon, J.A. Biological applications of electrostatic calculations and brownian dynamics simulations. *Rev. Computat. Chem.*, **1994**, *5*, 229–267.

[736] Gilson, M.K. Theory of electrostatic interactions in macromolecules. *Curr. Opin. Struct. Biol.*, **1995**, *5*, 216–23.

[737] Scarsi, M.; Apostolakis, J.; Caflisch, A. Continuum electrostatic energies of macromolecules in aqueous solutions. *J. Phys. Chem. A*, **1997**, *101*, 8098–8106.

[738] Simonson, T. Electrostatics and dynamics of proteins. *Rep. Prog. Phys.*, **2003**, *66*, 737–787.

[739] Bashford, D.; Karplus, M. p‘K sub a‘’s of ionizable groups in proteins: Atomic detail from a continuum electrostatic model. *Biochemistry*, **1990**, *29*, 10219–10225.

[740] Ghosh, A.; Rapp, C.S.; Friesner, R.A. Generalized Born model based on a surface integral formulation. *J. Phys. Chem. B*, **1998**, *102*, 10983–10990.

[741] Jackson, J.D. *Classical Electrodynamics*. Wiley and Sons, New York, 1975.

[742] Feig, M.; Karanicolas, J.; Brooks, C.L. III. MMTSB Tool Set: Enhanced sampling and multiscale modeling methods for application in structural biology. *J. Mol. Graphics Mod.*, **2004**, *22*, 377–395.

[743] Simmerling, C.; Strockbine, B.; Roitberg, A.E. All-atom structure prediction and folding simulations of a stable protein. *J. Am. Chem. Soc.*, **2002**, *124*, 11258–11259.

[744] García, A.E.; Sanbonmatsu, K.Y. α-helical stabilization by side chain shielding of backbone hydrogen bonds. *Proc. Natl. Acad. Sci. USA*, **2002**, *99*, 2782–2787.

[745] Kirschner, K.N.; Woods, R.J. Quantum mechanical study of the nonbonded forces in water-methanol complexes. *J. Phys. Chem. A*, **2001**, *105*, 4150–4155.

[746] Sharp, K.A.; Honig, B. Electrostatic interactions in macromolecules: Theory and experiment. *Annu. Rev. Biophys. Biophys. Chem.*, **1990**, *19*, 301–332.

[747] Gao, J. Absolute free energy of solvation from Monte Carlo simulations using combined quantum and molecular mechanical potentials. *J. Phys. Chem.*, **1992**, *96*, 537–540.

[748] Warshel, A.; Levitt, M. Theoretical studies of enzymic reactions: Dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme. *J. Mol. Biol.*, **1976**, *103*, 227–249.

[749] Field, M.J.; Bash, P.A.; Karplus, M. A combined quantum mechanical and molecular mechanical potential for molecular dynamics simulations. *J. Comput. Chem.*, **1990**, *11*, 700–733.

[750] Stanton, R.V.; Hartsough, D.S.; Merz, K.M. Jr. An examination of a density functional/molecular mechanical coupled potential. *J. Comput. Chem.*, **1994**, *16*, 113–128.

[751] Stanton, R.V.; Little, L.R.; Merz, K.M. Jr. An examination of a Hartree-Fock/molecular mechanical coupled potential. *J. Phys. Chem.*, **1995**, *99*, 17344–17348.

[752] Stanton, R.V.; Hartsough, D.S.; Merz, K.M. Jr. Calculations of solvation free energies using a density functional/molecular dynamics coupled potential. *J. Phys. Chem.*, **1993**, *97*, 11868–11870.

[753] Nocedal, J.; Wright, S.J. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[754] Nash, S.G. A survey of truncated-Newton methods. *J. of Computational and Applied Mathematics*, **2000**, *124*, 45–59.

[755] Sorin, E.J.; Pande, V.S. Exploring the helix-coil transition via all-atom equilibrium ensemble simulations. *Biophys. J.*, **2005**, *88*, 2472–2493.

[756] Rizzo, R.C.; Aynechi, T.; Case, D.A.; Kuntz, I.D. Estimation of absolute free energies of hydration using continuum methods: Accuracy of partial charge models and optimization of nonpolar contributions. *J. Chem. Theory Comput.*, **2006**, *2*, 128–139.

[757] Messiah, A. *Quantum Mechanics*. Wiley & Sons, New York, 1958.

[758] Storer, J.W.; Giesen, D.J.; Cramer, C.J.; Truhlar, D.G. Class IV charge models: A new semiempirical approach in quantum chemistry. *J. Comput.-Aided Mol. Design*, **1995**, *9*, 87–110.

[759] Li, J.; Cramer, C.J.; Truhlar, D.G. New class IV charge model for extracting accurate partial charges from Wave Functions. *J. Phys. Chem. A.*, **1998**, *102*, 1820–1831.

[760] van der Vaart, A.; Merz, K.M. Jr. Divide and conquer interaction energy decomposition. *J. Phys. Chem. A*, **1999**, *103*, 3321–3329.

[761] Mitin, A.V. The dynamic level shift method for improving the convergence of the SCF procedure. *J. Comput. Chem.*, **1988**, *9*, 107–110.

[762] Ermolaeva, M.D.; van der Vaart, A.; Merz, K.M. Jr. Implementation and testing of a frozen density matrix - divide and conquer algorithm. *J. Phys. Chem.*, **1999**, *103*, 1868–1875.

[763] Wang, B.; Brothers, E.N.; van der Vaart, A.; Merz Jr., K.M. Fast semiempirical calculations for nuclear magnetic resonance chemical shifts: A divide-and-conquer approach. *J. Chem. Phys.*, **2004**, *120*, 11392–11400.

[764] Wang, B.; Raha, K.; Merz Jr., K.M. Pose scoring by NMR. *J. Am. Chem. Soc.*, **2004**, *126*, 11430–11431.

[765] Raha, K.; van der Vaart, A.; E. Riley, K.; B. Peters, M.; M. Westerhoff, L.; Kim, H.; Merz Jr., K.M. Pairwise decomposition of residue interaction energies using semiempirical quantum mechanical methods in studies of protein-ligand interaction. *J. Am. Chem. Soc.*, **2005**, *127*, 6583–6594.

[766] Luzhkov, A.; Warshel, A. Microscopic models for quantum-mechanical calculations of chemical processes in solutions - Ld/Ampac and Scaas/Ampac calculations of solvation energies. *J. Comp. Chem.*, **1992**, *13*, 199–213.

[767] Singh, U.C.; Kollman, P.A. A combined Ab initio quantum-mechanical and molecular mechanical method for carrying out simulations on complex molecular systems - Applications to the Ch3Cl + Cl- exchange-reaction and gas-phase protonation of polyethers. *J. Comp. Chem.*, **1986**, *7*, 718–730.

[768] Bersuker, I.B.; Leong, M.K.; Boggs, J.E.; Pearlman, R.S. A method of combined quantum mechanical (QM) molecular mechanics (MM) treatment of large polyatomic systems with charge transfer between the QM and MM fragments. *Int. J. Quant. Chem.*, **1997**, *63*, 1051–1063.

[769] Maseras, F.; Morokuma, K. Imomm - a new integrated ab-initio plus molecular geometry optimization scheme of equilibrium structures and transition-states. *J. Comp. Chem.*, **1995**, *16*, 1170–1179.

[770] Zhang, Y.K.; Lee, T.S.; Yang, W.T. A pseudobond approach to combining quantum mechanical and molecular mechanical methods. *J. Chem. Phys.*, **1999**, *110*, 46–54.

[771] Gao, J.L.; Amara, P.; Alhambra, C.; Field, M.J. A generalized hybrid orbital (GHO) method for the treatment of boundary atoms in combined QM/MM calculations. *J Phys Chem A*, **1998**, *102*, 4714–4721.

[772] Philipp, D.M.; Friesner, R.A. Mixed ab initio QM/MM modeling using frozen orbitals and tests with alanine dipeptide and tetrapeptide. *J. Comp. Chem.*, **1999**, *20*, 1468–1494.

[773] Field, M.J.; Albe, M.; Bret, C.; Proust-De Martin, F.; Thomas, A. The Dynamo library for molecular simulations using hybrid quantum mechanical and molecular mechanical potentials. *J. Comp. Chem.*, **2000**, *21*, 1088–1100.

[774] Levy, R.M.; Gallicchio, E. Computer simulations with explicit solvent: recent progress in the thermodynamic decomposition of free energies and in modeling electrostatic effects. *Annu. Rev. Phys. Chem.*, **1999**, *49*, 531–567.

[775] Hornak, V.; Okur, A.; Rizzo, R.; Simmerling, C. HIV-1 protease flaps spontaneously open and reclose in molecular dynamics simulations. *Proc. Nat. Acad. Sci. USA*, **2006**, *103*, 915–920.

[776] Hornak, V.; Okur, A.; Rizzo, R.; Simmerling, C. HIV-1 protease flaps spontaneously close when an inhibitor binds to the open state. *J. Am. Chem. Soc.*, **2006**, *128*, 2812–2813.

[777] Ren, P.Y.; Ponder, J.W. Polarizable atomic multipole water model for molecular mechanics simulation. *J. Phys. Chem. B*, **2003**, *107*, 5933–5947.

[778] Ren, P.Y.; Ponder, J.W. Tinker polarizable atomic multipole force field for proteins. *to be published.*, **2006**.

[779] Simmerling, C.; Elber, R. Hydrophobic "collapse" in a cyclic hexapeptide: Computer simulations of CHDLFC and CAAAAC in water. *J. Am. Chem. Soc.*, **1994**, *116*, 2534–2547.

[780] Deng, Y.; Roux, B. Calculation of standard binding free energies: Aromatic molecules in the T4 lysozyme L99A mutant. *J. Chem. Theor. Comput.*, **2006**, *2*, 1255–1273.

[781] Fulle, S.; Gohlke, H. Analyzing the flexibility of RNA structures by constraint counting. *Biophys. J.*, **2008**, *DOI:10.1529/biophysj.107.113415*.

[782] Gohlke, H.; Kuhn, L. A.; Case, D. A. Change in protein flexibility upon complex formation: Analysis of Ras-Raf using molecular dynamics and a molecular framework approach. *Proteins*, **2004**, *56*, 322–327.

[783] Ahmed, A.; Gohlke, H. Multiscale modeling of macromolecular conformational changes combining concepts from rigidity and elastic network theory. *Proteins*, **2006**, *63*, 1038–1051.

[784] Pettersen, E.F.; Goddard, T.D.; Huang, C.C.; Couch, G.S.; Greenblatt, D.M.; Meng, E.C.; Ferrin, T.E. UCSF Chimera - A visualization system for exploratory research and analysis. *J. Comput. Chem.*, **2004**, *25*, 1605–1612.

[785] Sasaki, H.; Ochi, N.; Del, A.; Fukuda, M. Site-specific glycosylation of human recombinant erythropoietin: Analysis of glycopeptides or peptides at each glycosylation site by fast atom bombardment mass spectrometry. *Biochemistry*, **1988**, *27*, 8618–8626.

[786] Dube, S.; Fisher, J.W.; Powell, J.S. Glycosylation at specific sites of erythropoietin is essential for biosynthesis, secretion, and biological function. *J. Biol. Chem.*, **1988**, *263*, 17516–17521.

[787] Darling, R.J.; Kuchibhotla, U.; Glaesner, W.; Micanovic, R.; Witcher, D.R.; Beals, J.M. Glycosylation of erythropoietin effects receptor binding kinetics: Role of electrostatic interactions. *Biochemistry*, **2002**, *41*, 14524–14531.

[788] Cheetham, J.C.; Smith, D.M.; Aoki, K.H.; Stevenson, J.L.; Hoeffel, T.J.; Syed, R.S.; Egrie, J.; Harvey, T.S. NMR structure of human erythropoietin and a comparison with its receptor bound conformation. *Nat. Struct. Biol.*, **1998**, *5*, 861–866.

[789] Dormann, K.L.; Brueckner, R. Variable Synthesis of the Optically Active Thiotetronic Acid Antibiotics Thiolactomycin, Thiotetromycin, and 834-B1. *Angew. Chem. Int. Ed.*, **2007**, *46*, 1160–1163.

[790] Darve, Eric; Pohorille, Andrew. Calculating free energies using average force. *J. Chem. Phys.*, **2001**, *115*, 9169–9183.

[791] Shao, J.; Tanner, S.W.; Thompson, N.; Cheatham, T.E. III. Clustering molecular dynamics trajectories: 1. Characterizing the performance of different clustering algorithms. *J. Chem. Theory Comput.*, **2007**, *3*, 2312–2334.

[792] Brooks, B.R.; Bruccoleri, R.E.; Olafson, D.J.; States, D.J.; Swaminathan, S.; Karplus, M. CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations. *J. Computat. Chem.*, **1983**, *4*, 187–217.

[793] Brooks, B. R.; Brooks, C. L.; Mackerell, A. D.; Nilsson, L.; Petrella, R. J.; Roux, B.; Won, Y.; Archontis, G.; Bartels, C.; Boresch, S.; Caflisch, A.; Caves, L.; Cui, Q.; Dinner, A. R.; Feig, M.; Fischer, S.; Gao, J.; Hodoscek, M.; Im, W.; Kuczera, K.; Lazaridis, T.; Ma, J.; Ovchinnikov, V.; Paci, E.; Pastor, R. W.; Post, C. B.; Pu, J. Z.; Schaefer, M.; Tidor, B.; Venable, R. M.; Woodcock, H. L.; Wu, X.; Yang, W.; York, D. M.; Karplus, M. CHARMM: the biomolecular simulation program. *J. Comput. Chem.*, **2009**, *30*, 1545–1614.

[794] Ponder, J.W.; Wu, C.; Ren, P.; Pande, V.S.; Chodera, J.D.; Schieders, M.J.; Haque, I.; Mobley, D.L.; Lambrecht, D.S.; DiStasio, R.A. Jr.; Head-Gordon, M.; Clark, G.N.I.; Johnson, M.E.; Head-Gordon, T. Current status of the AMOEBA polarizable force field. *J. Phys. Chem. B*, **2010**, *114*, 2549–2564.

[795] Hsieh, M.-J.; Luo, R. Balancing simulation accuracy and efficiency with the Amber united atom force field. *J. Phys. Chem. B*, **2010**, *114*, 2886–2893.

[796] Hirata, F., Ed. *Molecular Theory of Solvation*. Kluwer Academic Publishers, 2003.

[797] Li, D.W.; Brüschweiler, R. NMR-based protein potentials. *Angew. Chem. Int. Ed.*, **2010**, *49*, 6778–6780.

[798] Lindorff-Larsen, K.; Piana, S.; Palmo, K.; Maragakis, P.; Klepeis, J.; Dror, R.O.; Shaw, D.E. Improved side-chain torsion potentials for the Amber ff99SB protein force field. *Proteins*, **2010**, *78*, 1950–1958.

[799] Cerutti, D.S.; Duke, R.E.; Darden, T.A.; Lybrand, T.P. Staggered Mesh Ewald: An Extension of the Smooth Particle-Mesh Ewald Method Adding Great Versatility. *J. Chem. Theory Computat.*, **2009**, *5*, 2322–2338.

[800] Cerutti, D.S.; Freddolino, P.L.; Duke, R.E. Jr.; Case, D.A. Simulations of a Protein Crystal with a High Resolution X-ray Structure: Evaluation of Force Fields and Water Models . *J. Phys. Chem. B*, **2010**, pp 12811–12824.

[801] Gaillard, T.; Case, D.A. Evaluation of DNA Force Fields in Implicit Solvation. *J. Chem. Theory Comput.*, **2011**, *7*, 3181–3198.

[802] Meng, Yilin; Roitberg, Adrian E. Constant pH replica exchange molecular dynamics in biomolecules using a discrete protonation model. *J. Chem. Theory Comput.*, **2010**, *6*, 1401–1412.

[803] Sabri Dashti, Danial; Meng, Yilin; Roitberg, Adrian E. pH-Replica Exchange Molecular Dynamics in Proteins Using a Discrete Protonation Method. *J. Phys. Chem. B*, **2012**, *116*, 8805–8811.

[804] Wu, X.; Brooks, B. R. A virtual mixture approach to the study of multistate equilibrium: application to constant ph simulation in explicit water. *PLOS Computational Biology*, **2015**, *11*, e1004480.

[805] Karamertzanis, P.G.; Raiteri, P.; Galindo, A. The use of anisotropic potentials in modeling water and free energies of hydration. *J. Chem. Theory Comput.*, **2010**, *6*, 3153–3161.

[806] Wang, J.; Cieplak, P.; Li, J.; Hou, T.; Luo, R.; Duan, Y. Development of Polarizable Models for Molecular Mechanical Calculations I: Parameterization of Atomic Polarizability. *J. Phys. Chem. B*, **2011**, *115*, 3091–3099.

[807] Wang, J.; Cieplak, P.; Li, J.; Wang, J.; Cai, Q.; Hsieh, M.; Lei, H.; Luo, R.; Duan, Y. Development of Polarizable Models for Molecular Mechanical Calculations II: Induced Dipole Models Significantly Improve Accuracy of Intermolecular Interaction Energies. *J. Phys. Chem. B*, **2011**, *115*, 3100–3111.

[808] Wang, J.; Cieplak, P.; Li, J.; Cai, Q.; Hsieh, M.; Luo, R.; Duan, Y. Development of Polarizable Models for Molecular Mechanical Calculations. 4. van der Waals Parametrization. *J. Phys. Chem. B*, **2012**, *116*, 7088–7101.

[809] Wang, J.; Cieplak, P.; Cai, Q.; Hsieh, M.; Wang, J.; Duan, Y.; Luo, R. Development of Polarizable Models for Molecular Mechanical Calculations. 3. Polarizable Water Models Conforming to Thole Polarization Screening Schemes. *J. Phys. Chem. B*, **2012**, *116*, 7999–8008.

[810] Thole, B.T. Molecular polarizabilities calculated with a modified dipole interaction. *Chem. Phys.*, **1981**, *59*, 341–350.

[811] Bosque, R.; Sales, J. Polarizabilities of solvents from the chemical composition. *J. Chem. Inf. Comput. Sci.*, **2002**, *42*, 1154–1163.

[812] Wang, Z.X.; Wu, C.; Lei, H.X.; Duan, Y. Accurate ab initio study on the hydrogen-bond pairs in protein secondary structures. *J. Chem. Theory Comput.*, **2007**, *3*, 1527–1537.

[813] Wang, J.; Hou, T. Application of Molecular Dynamics Simulations in Molecular Property Prediction. II. Diffusion coefficient. *J. Comput. Chem.*, **2011**, *32*, 3509–3519.

[814] Fu, C.F.; Tian, S.X. A Comparative Study for Molecular Dynamics Simulations of Liquid Benzene. *J. Chem. Theory Comput.*, **2011**, *7*, 2240–2252.

[815] Tsuzuki, S.; Uchimaru, T.; Tanabe, K.; Kuwajima, S. Refinement of Nonbonding Interaction Potential Parameters for Methane on the Basis of the Pair Potential Obtained by Mp3/6-311g(3d,3p)-Level Ab-Initio Molecular-Orbital Calculations - the Anisotropy of H/H Interaction. *J. Phys. Chem.*, **1994**, *98*, 1830–1833.

[816] Kaminski, G.A.; Friesner, R.A.; Tirado-Rives, J.; Jorgensen, W.L. Evaluation and Reparametrization of the OPLS-AA Force Field for Proteins via Comparison with Accurate Quantum Chemical Calculations on Peptides. *J. Phys. Chem. B*, **2001**, *105*, 6474–6487.

[817] Chen, I.J.; Yin, D.; MacKerell, A.D. Combined Ab initio/Empirical Approach for Optimization of Lennard-Jones Parameters for Polar-Neutral Compounds. *J. Comput. Chem.*, **2002**, *23*, 199–213.

[818] Dupradeau, F.-Y.; Pigache, A.; Zaffran, T.; Savineau, C.; Lelong, R.; Grivel, N.; Lelong, D.; Rosanskia, W.; Cieplak, P. The R.E.D. tools: advances in RESP and ESP charge derivation and force field library building. *PhysChemChemPhys*, **2010**, *12*, 7821–7839.

[819] Warschawski, D.E.; Devaux, P.F. Order parameters of unsaturated phospholipids in membranes and the effect of cholesterol: a 1H-13C solid-state NMR study at natural abundance. *Eur. Biophys. J.*, **2005**, *34*, 987–996.

[820] Showalter, S.A.; Brüschweiler, R. Validation of molecular dynamics simulations of biomolecules using NMR spin relaxation as benchmarks: Application to the Amber99SB force field. *J. Chem. Theory Comput.*, **2007**, *3*, 961–975.

[821] Best, R.B.; Buchete, N.-V.; Hummer, G. Are Current Molecular Dynamics Force Fields too Helical? *Biophys. J.*, **2008**, *95*, L07–L09; 4494.

[822] Best, R.B.; Hummer, G. Optimized Molecular Dynamics Force Fields Applied to the Helix-Coil Transition of Polypeptides. *J. Phys. Chem. B*, **2009**, *113*, 9904–9015.

[823] Best, R.B.; Mittal, J. Free-energy landscape of the GB1 hairpin in all-atom explicit solvent simulations with different force fields: Similarities and differences. *Proteins*, **2011**, *79*, 1318–1328.

[824] Patapati, K.K.; Glykos, N.M. Three force fields views of the 3-10 helix. *Biophys. J.*, **2011**, *101*, 1766–1771.

[825] Salomon-Ferrer, R.; Case, D.A.; Walker, R.C. An overview of the Amber biomolecular simulation package. *WIREs Comput. Mol. Sci.*, **2013**, *3*, 198–210.

[826] PDB Current Holdings Breakdown. **2013**.

[827] Nguyen, Crystal N.; Kurtzman Young, Tom; Gilson, Mike K. Grid Inhomogeneous solvation theory: Hydration structure and thermodynamics of the miniature receptor cucurbit[7]uril. *J. Chem. Phys.*, **2012**, *137*, 044101–044118.

[828] Fernandez, D. P.; Goodwin, A. R. H.; Lemmon, Eric W.; Levelt Sengers, J. M. H.; Williams, R. C. A formulation for the static permittivity of water and steam at temperatures from 238 k to 873 k at pressures up to 1200 mpa, including derivatives and debye huckel coefficients. *J. Phys. Chem. Ref. Data*, **1997**, *26*, 1125–1166.

[829] Mills, R. Self-diffusion in normal and heavy water in the range 1-45. deg. *J. Phys. Chem.*, **1973**, *77*, 685–688.

[830] Wagner, W.; Pruss, A. The iapws formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use. *J. Phys. Chem. Ref. Data*, **2002**, *31*, 387–535.

[831] Skinner, Lawrie B.; Huang, Congcong; Schlesinger, Daniel; Pettersson, Lars G. M.; Nilsson, Anders; Benmore, Chris J. Benchmark oxygen-oxygen pair-distribution function of ambient water from x-ray diffraction measurements with a wide q-range. *J. Chem. Phys.*, **2013**, *138*, 074506+.

[832] Kell, G. S. Precise representation of volume properties of water at one atmosphere. *J. Chem. Eng. Data*, **1967**, *12*, 66–69.

[833] Vega, Carlos; Abascal, Jose L. F. Simulating water with rigid non-polarizable models: a general perspective. *Phys Chem Chem Phys*, **2011**, *13*, 19663–19688.

[834] Ruscio, Jory Z.; Kumar, Deept; Shukla, Maulik; Prisant, Michael G.; Murali, T. M.; Onufriev, Alexey V. Atomic level computational identification of ligand migration pathways between solvent and binding site in myoglobin. *Proc. Nat. Acad. Sci. USA*, **2008**, *105*, 9204–9209.

Index

1D-RISM, 94, 95, 102
3D-RISM, 94, 96, 105, 110

A
abfqmmm, 179, 348
accept, 75, 822
acdoctor, 290
acos, 793
activeref, 569
adbcor, 104
add, 216
add_12_6_4, 253
add_pdb, 252
addAtomicNumber, 251
addAtomTypes, 217
addDihedral, 251
addExclusions, 252
addIons, 217
addIons2, 217
addIonsRand, 218
addLJType, 252
addPath, 218
addPDB, 252
addPdbAtomMap, 218
addPdbResMap, 218
addresidue, 763, 795
addstrand, 763, 795
adjust_q, 348
adjust_q, 154
aexp, 480
alias, 219
alignframe, 769, 802
allatom_to_dna3, 797
allocate, 782
alpb, 62, 344
alpha, 411
am1bcc, 287
anchor_postion, 445
anchor_strength, 445
andbounds, 810
angle, 590, 798
anglep, 798
antechamber, 279
apply_rism_force, 825
apply_rism_force, 114
arad, 63
arange, 480
arcres, 74, 822
arnoldi_dimension, 403
asin, 793
assert, 800
asymp, 108
asympcorr, 113
asympfile, 823
atan, 793
atan2, 793
atmask, 500
atnam, 476
atof, 793
atoi, 793
atomicfluct, 591
atommap, 592
atomn, 338
atomtype, 286
auxbasis, 165
average, 593
awt, 480

B
bar_intervall, 415
bar_l_incr, 415
bar_l_max, 415
bar_l_min, 415
barostat, 323
basepair, 769
basis, 160, 161, 163–166, 168
bcopt, 76, 822
bdna, 769, 848
bdna(), 770
beckegrid, 160
bellymask, 319
blocksize, 827
bond, 219
bondByDistance, 219
bondtype, 287
bond_umb, 135
break, 788
bridge function, 95
buffer, 108, 113, 694, 824
buffer_iqmatoms, 348

buffercharge, 179, 347
buffermask, 179, 348

C
calc, 166
calc_wbk, 174
car_to_files.py, 306
CartHess2FC.py, 303
cavity_offset, 692
cavity_surften, 692
cbasis, 164
ccut, 486
ceil, 793
center, 595
center_type, 180
centering, 109, 114, 825
centermask, 180, 348
chamber, 254
change, 261
changeLJ14Pair, 261
changeLJPair, 261
changeLJSingleType, 261
changeProtState, 262
changeRadii, 262
charge, 183
charge density, 112
charge distribution, 112
charge_analysis, 169
charge_out, 186
check, 220, 595
checkValidity, 262
chelpg, 162
chgdist, 108, 823
chkvir, 338
chngmask, 332
clambda, 407, 415
closest, 596
closure, 101, 102, 108, 112, 694, 823
closureorder, 108, 694, 823
cluster, 652
clusterdihedral, 597
cobsl, 486
column_fft, 330
combine, 220
comp, 324
complement, 769
conflib_filename, 403
conflib_size, 403
conjgrad, 818
connectres, 763, 795
continue, 788
convkey, 164
convthre, 169
copy, 220
copymolecule, 795
core, 160
core_iqmatoms, 348
corecharge, 179, 347
coremask, 179, 348
corembed, 167
corembedatoms, 167
corr, 658
correlation, 165
cos, 793
cosh, 793
countmolatoms, 798
cphstats, 466
create, 569
createAtom, 221
createResidue, 221
createUnit, 221
crgmask, 411
csurften, 324
csv_format, 693
cter, 482
cut, 327, 344, 819
cut_bond_list_file, 181
cutcap, 326
cutfd, 77
cutnb, 77, 822
cuv, 108, 111, 112
cuvfile, 823
cv_file, 445
cv_i, 441
cv_max, 448
cv_min, 448
cv_ni, 442
cv_nr, 442
cv_r, 442
cv_type, 441
cwt, 486

D
damp, 346
datafile, 569
dataset, 485
datasetc, 486
date, 801
dbfopt, 822
dbonds_umb, 135
dbuff1, 184
dbuff2, 184
dcut, 485
deallocate, 782
debug, 167, 571, 800
debug_printlevel, 690

dec_verbose, 693
decompopt, 77
defineSolvent, 262
delete, 786
deleteBond, 221, 262
deleteDihedral, 262
deletePDB, 262
density, 103
density_predict, 348
desc, 221, 436
dft, 166
dftb_3rd_order, 153
dftb_3rd_order, 141, 349
dftb_chg, 141, 348
dftb_disper, 141, 348
dftb_maxiter, 141, 348
dftb_slko_path, 141
dftb_telec, 141, 346
dftb_telec_step, 346
dftb_chg, 153
dftb_maxiter, 153
dftb_telec, 153
dftd, 169
dftgrid, 169
dg_helix, 848
dg_options, 810
dgpt_alpha, 136
diag_routine, 348
diag_routine, 142, 153
dia_shift, 134
diel, 820
dielc, 327, 344, 693, 820
dieps, 104
dihedral, 601
dij, 485
dim, 819
dipmass, 332
dipole, 160, 162–164, 166, 169
diptau, 332
diptol, 332
direct correlation function, 94
dist, 798
distance, 602
dist_gauss, 136
distp, 798
dna3, 797
dna3_to_allatom, 797
do_dipole, 167
dobsl, 485
do_debugf, 338
dpmax, 185
dprob, 73, 822
dr, 103
drms, 320, 402, 693
drmsd, 602
dsum_tol, 329
dt, 320
dtfb_disper, 153
dumpatom, 800
dumpbounds, 800
dumpboundsviolations, 800
dumpfrc, 338
dumpmatrix, 800
dumpmolecule, 800
dumpresidue, 800
dvbips, 331
dvdl_norest, 411
dwt, 485
DX, 122
dx0, 320
dynamicgrid, 169
dynlmb, 411

E
e_debug, 819
eedmeth, 330
eedtbdns, 330
effreq, 329
efphase, 329
efx, 328
efy, 328
efz, 328
egap_umb, 135
emap, 135
embed, 167, 810
embedatoms, 167
emem, 692
EMIL, 415
emil_do_calc, 415
emil_logfile, 416, 418
emil_model_infile, 416
emil_model_outfile, 416
emil_paramfile, 416
emil_sc, 415
emix, 480
endframe, 690
ene_avg_sampling, 367
eneopt, 76, 822
energy, 263
energy_window, 403
entropicDecomp, 102, 109, 117, 826
entropy, 690
epsext, 821
epsin, 73, 821
epsmem, 73
epsout, 73, 821

eq_cmd, 517
equilibration, 452
errconv, 144, 346
es_cutoff, 366
espgen, 288
espgen.py, 305
evb_dyn, 133
ew_type, 329, 344
ew_coeff, 330
exactdensity, 160
excess chemical potential, 97
exch_type, 436
exchange, 165
exchange_freq, 450
exchange_log_file, 450
exchange_log_freq, 450
exdi, 692
executable, 169
exit, 792
exp, 793
explored_low_modes, 403
ext_buffermask_subset, 181
ext_coremask_subset, 181
ext_qmmask_subset, 181
extdiel, 61, 343
extra_precision, 104

F
fabs, 793
fcap, 326
FCE, 110
fcecrd, 115
fceenormsw, 115
fceifreq, 116
fcenbase, 115
fcenbasis, 115
fcentfrcor, 116
fcesort, 115
fcestride, 115
fcetrans, 115
fceweigh, 115
fclose, 792
fcons, 500
fd_helix, 796
fft_grids_per_ang, 367
fillratio, 75, 692, 822
finalgrid, 164
fit_type, 160
fix_atom_list, 180
fixcom, 388
floor, 793
fmod, 793
fock_predict, 348
fockp_d1, 346
fockp_d2, 346
fockp_d3, 346
fockp_d4, 346
fopen, 792
fprintf, 792
frameon, 332
frcopt, 76, 822
freemolecule, 795
freeresidue, 795
freezemol, 485
frequency_eigenvector_recalc, 403
frequency_ligand_rotrans, 403
fscale, 75, 822
fscanf, 792
fswitch, 328
ftime, 801
full_traj, 691
fullscf, 185
fxyz, 479

G
gamma_ln, 179
gamma_ln_qm, 180
gamma_ten, 324
gamma_ln, 322, 820
gammamap, 326
gauss, 793
Gaussian fluctuation, 97, 109, 826
gb, 821
gb2_debug, 819
gb_debug, 819
gbsa, 62, 344, 821
genmass, 820
geodesics, 810
getchivol, 810
getchivolp, 810
getcif, 798
getline, 792
getmatrix, 794
getpdb, 798
getpdb_prm, 818
getres, 764, 769
getresidue, 763, 764, 798
getxv, 818
gfCorrection, 112
gigj, 485
gist, 605
gms_version, 162
go, 264
gpu, 373
gpuids, 169
grdspc, 108, 113, 694, 824

grid, 164
gridips, 331
grids, 500
grms_tol, 143
grnam1, 479
gromber, 264
group, 435
groupSelectedAtoms, 222
gsub, 791
guess, 166
guv, 108, 111
guvfile, 823

H
hamiltonian, 184
harm, 445
harm_mode, 445
hbond, 613
hcp, 827
helix, 769, 770
helixanal, 800
help, 572
hist, 662
history, 265
HMassRepartition, 264
HNC, 95, 97
hot_spot, 182, 348
huv, 108, 111
huvfile, 823
hybridgb, 427
hypernetted-chain approximation, 95

I
ialtd, 477
iamoeba, 328
iat, 475
iatr, 482
ibelly, 319
icfe, 407, 415
iconstr, 479
icsa, 486
id, 485
id2o, 481
idecomp, 319, 408, 694
idistr, 323
iemap, 326
ievb, 125, 328
ifit, 500
ifmbar, 415
ifntyp, 479
ifqnt, 328, 344, 691
ifsc, 411, 415
ifvari, 477, 478
ig, 322
igb, 59, 263, 267, 328, 344, 691
igr1, 479
ihp, 480
image, 452, 615
imin, 72, 316
impose, 223
imult, 477
index, 791
indi, 692
indmeth, 332
ineb, 400
infe, 441
initial_selection_type, 180
inp, 72, 692, 821
intdiel, 61, 343
integration, 160
interpolate, 265
interval, 690
intramolecular pair correlation matrix, 95
invwt1, 480
ioutfm, 318
ipb, 72, 344, 821
ipimd, 514, 516, 518
IPMach.py, 301
ipnlty, 326
ipol, 328
ipolyn, 185
iprob, 73, 822
iprot, 482, 483
ips, 331
iqmatoms, 151, 346
ir6, 479
iresid, 477
irest, 317
irism, 112, 328, 823
irstdip, 332
irstyp, 477
iscale, 326
isgend, 388
isgld, 388
isgsta, 388
istrng, 73, 692, 821
itgtmd, 396
itrmax, 143, 153, 347
ivcap, 325
iwrap, 318
ixpk, 479

J
jbasis, 164
jcoupling, 616
jfastw, 325, 344

K
k4d, 819
kappa, 346, 821
keep_files, 690
KH, 95, 97
klambda, 407, 415
kmaxqx, 152, 347
kmaxqy, 348
kmaxqz, 348
Kovalenko-Hirata, 95
ksave, 103
ksqmaxq, 152
ksqmaxsq, 348

L
lambda, 185, 415
lbfgs_memory_depth, 402
length, 791
lie, 617
ligand_mask, 690
ligcent_list, 404
ligstart_list, 404
linit, 692
link_atomic_no, 347
link_na, 796
linkprot, 796
list, 224
listParms, 265
lj1264, 328, 344
lmod, 839
lmod_job_title, 403
lmod_minimize_grms, 403
lmod_relax_grms, 403
lmod_restart_frequency, 403
lmod_step_size_max, 403
lmod_step_size_min, 403
lmod_trajectory_filename, 403
lmod_verbosity, 404
lnk_dis, 345
lnk_method, 347
lnk_atomic_no, 154
lnk_dis, 154
lnk_method, 154
loadAmberParams, 225
loadAmberPrep, 225
loadCoordinates, 266
loadMol2, 225
loadOff, 225
loadPdb, 225
loadPdbUsingSeq, 226
loadRestrt, 266
log, 793
log10, 793
logdvdl, 411
logFile, 226
longrange, 185
long-range asymptotics, 112

M
mapfile, 500
mapfit, 500
mask, 620
match, 292, 791
match_atomname, 292
MAT_cube, etc, 803
matextract, 807
MAT_fprint, etc, 804
matgen, 805
matmerge, 807
matrix_vector_product_method, 402
max_bonds_per_atom, 181
maxarcdot, 822
maxcore, 164
maxcyc, 143, 320, 402, 693
maxit, 162, 169
maxiter, 164, 332
maxitn, 75, 822
maxsph, 78
maxstep, 104, 109, 114, 825
mcbarint, 324
MCPB.py, 296
mctrdz, 74, 693
md, 818
MDIIS, 96, 101
mdiis_del, 101, 104, 109, 113, 824
mdiis_method, 114, 824
mdiis_nvec, 101, 104, 109, 113, 824
mdiis_restart, 101, 104, 109, 114, 824
mdinfo_flush_interval, 366
MDL, 118
mdout_flush_interval, 366
mean solvation force, 97
measureGeom, 226
mem, 163, 166
membraneopt, 74, 80
memopt, 692
mergestr, 763, 795
method, 161, 163–165, 169, 183
min_heavy_mass, 181, 346
minimize, 266
min_xfile, 136
mipso, 331
mipsx, 331
mlimit, 330
mltpro, 483
mme, 818

mme_rism_max_memory, 801
mme_timer, 801
mme_init, 818
mme_rattle, 818
mm_options, 818
mm_set_checkpoint, 818
mode, 448
model, 103
modif, 184
modified direct inversion of the iterative subspace, 96
modvdw, 135
molfit, 501
molsurf, 622, 692, 800
mom_cons_region, 180
mom_cons_type, 180
monitor_file, 448
monitor_freq, 448
monte_carlo_method, 404
morsify, 134
move, 500
MPI, 760
mprob, 74
msoffset, 692
mt19937_file, 450
mt19937_seed, 450
mthick, 74, 80, 692
mtmdforce, 398
mtmdform, 397
mtmdmask, 398
mtmdmult, 398
mtmdninc, 398
mtmdrmsd, 398
mtmdstep1, 397
mtmdvari, 397
MTS, 110
multi-dimensional replica exchange, 434
multipmemd, 340
multisander, 340
mutant_only, 693
mwords, 162
mxsub, 326

N
n_max_recursive, 181
n_partition, 174
NAB, 105, 107
namr, 482
nastruct, 624
natr, 482
nbflag, 330
nbias, 133
nbtell, 330
nbuffer, 75
nchain, 179, 516
nchk, 819
nchk2, 819
ncore, 184
ncyc, 320
ndiis_attempts, 347
ndiis_matrices, 347
ndiis_attempts, 143
ndiis_matrices, 144
ndip, 485, 486
nearest_qm_solvent, 174
nearest_qm_solvent_center_id, 174
nearest_qm_solvent_fq, 174
nearest_qm_solvent_resname, 174
neglgdel, 338
netcdf, 691
netCharge, 267
netfrc, 330
nevb, 132
newbounds, 810
newmolecule, 763, 795
newton, 832
newtransform, 768, 802
nfe_abmd, 448
nfe_bbmd, 450
nfe_pmd, 445
nfe_smd, 445
nfe_stsm, 450
nfft3, 329
nfocus, 75, 822
ng, 109, 694, 824
ng3, 113
ngpus, 169
ngrdblkx, 79
ngrdblky, 79
ngrdblkz, 79
nharm, 445
ninc, 477
ninterface, 324
nkija, 323
nkout, 103
nleb, 161
nme, 483
nmendframe, 693
nminterval, 693
nmo_corembed, 167
nmo_embed, 167
nmode, 832
nmode_igb, 693
nmode_istrng, 693
nmodvdw, 133
nmorse, 132
nmpmc, 483

nmropt, 317
nmstartframe, 693
noasympcorr, 108
noeskp, 326
noexitonerror, 572
no_intermolecular_bonds, 366
noprogress, 572
noshakemask, 325, 409
npath, 445
npbgrid, 75, 822
npbopt, 75, 822
npbverb, 77, 822
npeak, 480
nprintlog, 167
npropagate, 99, 109, 114, 825
nprot, 482, 483
nr, 103
nrad, 161
nranatm, 338
nrespa, 320
nring, 482
nrout, 103
nscm, 320, 518, 819
nsnb, 328, 819
nsnba, 77
nsp, 104
nstep1, 477
nstlim, 320
ntave, 318
ntb, 327, 344
ntc, 324, 344
nter, 482
ntf, 327, 344
ntmin, 320, 402
ntp, 323
ntpr, 143, 160, 162–164, 166, 167, 169, 317, 819
ntprism, 826
ntpr_md, 820
ntr, 319
ntt, 179, 321, 514, 516, 518
ntwc, 186
ntwe, 318
ntw_evb, 132
ntwf, 318
ntwidrst, 182
ntwpdb, 181
ntwprt, 319
ntwr, 318
ntwrism, 116, 826
ntwv, 318
ntwx, 318, 820
ntx, 72, 317
ntxo, 317
nuff, 133
num_mpi_prcs, 165
num_threads, 160, 162–165
number_free_rotrans_modes, 404
number_ligand_rotrans, 404
number_ligands, 404
number_lmod_iterations, 404
number_lmod_moves, 404
num_datasets, 485
numwatkeep, 427
nvec, 99
nxpk, 479

O
obs, 482, 483
offset, 62, 78
omega, 481
OMP_NUM_THREADS, 188, 760
OpenMM, 267
OptC4.py, 303
optkon, 483
optphi, 483
orbounds, 810
order, 329
Ornstein-Zernike, 94
oscale, 481
outCIF, 268
outlist, 103
outlvlset, 77
outmlvlset, 77
outparm, 268
output_file, 445
output_freq, 445
out_RCdot, 136
outtraj, 629
outxyz, 479
oxidation_number_list_file, 180

P
pair-distribution function, 94
param, 516
parameter_file, 153
parameterfle, 143
paramfit, 187
parm, 269, 577
parmbox, 578
parmcal, 291
parmchk2, 282
ParmEd, 250
parminfo, 579
parmout, 269
parmresinfo, 580
parmstrip, 579

parmwrite, 579
path, 445
path_mode, 445
PBRadii, 228
pbtemp, 73
PC+/3D-RISM, 109, 112, 826
pdb, 108
pdb_file, 181
PdbSearcher.py, 302
pencut, 326
peptcorr, 186
peptide_corr, 347
peptide_corr, 143, 153
peptk, 186
phiform, 77, 83
phiout, 83
pH-REMD, 432
plane, 800
point, 790
polarDecomp, 100, 109, 117, 826
polardecomp, 99, 694
poreradius, 74
poretype, 74, 693
pow, 793
prbrad, 692
precision, 169, 572
prepgen, 288
pres0, 324
print_eigenvalues, 143, 347
print_qm_coords, 175
print_res, 694
printAngles, 269
printbondorders, 347
printBonds, 269
printcharges, 143, 153, 347
printDetails, 270
printDihedrals, 270
printdipole, 153, 347
printf, 792
printFlags, 270
printInfo, 270
printLJMatrix, 270
printLJTypes, 270
printPointers, 270
prmtop, 108
probe, 692
profile_mpi, 333
progress, 103, 117, 826
ProScrs.py, 305
pseduo_diag_criteria, 346
PSE-n, 95, 97
pseudo_diag, 347
pseudo_diag, 142, 153
pseudo_diag_criteria, 142, 153
pucker, 632
putbnd, 798
putcif, 798
putdist, 798
putmatrix, 795
putpdb, 798
putxv, 818

Q
qm_center_atom_id, 174
qm_ewald, 347
qm_pme, 347
qm_residues, 691
qm_theory, 349, 691
qmcharge, 141, 153, 179, 347, 692
qmcut, 151, 345, 692
qm_ewald, 151
qmgb, 152, 346
qmmask, 151, 179, 348
qmmm_int, 348
qmmm_switch, 348
qmmm_int, 144, 152
qmmmrij_incore, 347
qmmm_switch, 152
qm_pme, 152
qmqm_erep_incore, 347
qmqmdx, 141, 153, 347
qmshake, 153, 347
qm_theory, 140, 152, 153
quit, 271
quv, 108
quvfile, 823
qxd, 143, 153

R
r0, 478
r_buffer_in, 179
r_core_in, 179
r_core_out, 179
r_qm_in, 179
r_switch_hi, 346
r_switch_lo, 346
RA, 174
radgyr, 633
radial, 633
radiopt, 73, 692, 821
raips, 331
rand2, 793
random_seed, 404
ranseed, 338
rattle, 820
rbornstat, 62

rdt, 62, 344
read_idrst_file, 181
readdata, 573
readinput, 573
readparm, 818
receptor_mask, 691
reference, 582
refin, 397
release, 452
remove, 227
repeats, 452
report_centers, 452
residuegen, 291
resolution, 448, 500
respgen, 289
restart_pool_size, 404
restraint, 476
restraintmask, 319
restraint_wt, 319
restrt_cmd, 517
rgbmax, 61, 343, 821
rhow_effect, 78
RISM, 94, 110
`rism1d`, 94, 100–102
rism3d.snglpnt, 108
rism_verbose, 694
`RISMnRESPA`, 110
rismnrespa, 115
rjcoef, 478
rms2d, 672
rmsavgcorr, 673
rmsd, 635, 798
rmsfrc, 338
rot4, 768, 802
rot4p, 768, 802
rotmin_list, 405
rseed, 793
rst, 108
Rst7, 361
rstwt, 475
rsum_tol, 330
r_switch_hi, 152
r_switch_lo, 152
RT, 174
rtemperature, 404
runavg, 638

S
s11, 485
saltcon, 61, 263, 267, 343, 692
sander, 110
sander_apbs, 692
saopt, 74
sasopt, 74
saveAmberParm, 227
saveMol2, 227
saveOff, 227
savePdb, 228
ScaLAPACK, 760
scaldip, 333
scale, 271, 692
scalec, 76
scalm, 326
scanf, 792
scee, 271
scf_conv, 160, 162, 163, 165
scf_iter, 160
scfconv, 142, 153, 164, 346
scmask, 411
scmask1, 412
scmask2, 412
scnb, 271
screen, 185
search_path, 691
secstruct, 638
select, 574
selection_constant, 449
selection_epsilon, 449
selection_freq, 448
selection_type, 180
selftest, 103
sequence, 228
set, 228
set container, 229
set default, 228
setAngle, 271
setBond, 271
setbounds, 810
setboundsfromdb, 810
setBox, 230
setchiplane, 810
setchivol, 810
setframe, 769, 802
setframep, 769, 802
setMolecules, 271
setmol_from_xyz, 803
setmol_from_xyzw, 803
setOverwrite, 272
setpoint, 803
setseed, 793
setxyz_from_mol, 803
setxyzw_from_mol, 803
sgff, 388
sgft, 388
shcut, 482
showbounds, 810

shrang, 482
sin, 793
sinh, 793
sinrtau, 323
skinnb, 330
skmax, 400
skmin, 400
smear, 104
smoothing, 452
smoothopt, 73, 821
snapshots_basename, 448
snapshots_freq, 448
solvateBox, 230
solvateCap, 231
solvateOct, 230
solvateShell, 231
solvation, 94, 97
solvation free energy, 97
solvbox, 109, 113, 694, 824
solvcut, 108, 113, 694, 823
solvent_atom_number, 180
solvopt, 75, 822
source, 272
space, 75, 822
spin, 141, 153, 347
split, 791
sprintf, 792
sprob, 78, 822
sqrt, 793
sscanf, 792
startframe, 691
static_arrays, 827
str, 482
strip, 272, 642
strip_mask, 691
sub, 791
substr, 791
sugarpuckeranal, 798
summary, 272
superimpose, 798
surf, 642
surfoff, 692
surften, 62, 78, 692, 821
system, 792

T
t, 320, 820
tan, 793
tanh, 793
taumet, 481
taup, 324
taurot, 481
tausw, 326
tautp, 322, 820
temp0, 322, 820
temp0les, 322
temperature, 104
Temperature REMD, 423
tempi, 322, 820
tempsg, 388
tgtfitmask, 396, 400
tgtmdfrc, 396
tgtrmsd, 396
tgtrmsmask, 396, 400
theory, 102
thermo, 695
thermodynamics, 97
threall, 169
tight_p_conv, 348
tight_p_conv, 142, 153
timask1, 408
timask2, 408
timeofday, 801
tiMerge, 273
timescale, 448
tishake, 407, 409, 411
tmode, 400
tol, 325
tolerance, 101, 104, 109, 113, 695, 824
tolpro, 483
torsion, 798
torsionp, 798
total correlation function, 94
total_low_modes, 404
trajin, 583
trajout, 584
trans4, 768
trans4p, 768
transform, 223, 231, 232, 808
transformmol, 768, 803
transformres, 763, 764, 768, 803
translate, 232
treflf, 388
triopt, 74
trmin_list, 405
tsgavg, 388
tsmooth, 810
ts_xfile, 136

U
uccoeff, 112
uff, 136
Universal Correction, 109, 826
unlink, 792
unstrip, 644
use_dftb, 160

use_sander, 691
use_template, 160, 162–165, 167, 169
use_axis_opt, 367
useboundsfrom, 810
use_rmin, 78
use_sav, 78
uuv, 108
uuvfile, 823

V
vdw_cutoff, 366
vdwmeth, 330, 344
verbose, 110, 117, 329, 691, 826
verbosity, 142, 153, 166, 174, 175, 232, 347
vfac, 400
vlimit, 323, 820
volfmt, 108, 116, 823
vprob, 78
vrand, 322
vshift, 144, 346
vsolv, 153, 348
vv, 400

W
wc_basepair, 848
wc_basepair(), 772
wc_complement, 848
wc_complement(), 770
wc_helix, 848
wc_helix(), 774
wcons, 819
write_idrst_file, 182
write_thermo, 117
writeFrcmod, 273
writeOFF, 273
writepdb, 153
wt, 482, 483
wt_temperature, 449
wt_umbrella_file, 449

X
xc, 160
xch_cnst, 134
xch_exp, 134
xch_gauss, 134
xch_type, 133
xdg_xfile, 136
xmax, 79
xmin, 79, 835
xmin_method, 402
xmin_verbosity, 402
XVV, 119
xvv, 108, 111
xvvfile, 823

Y
ymax, 79
ymin, 79

Z
zcap, 326
zerochg, 338
zerodip, 338
zerofrc, 114, 825
zerov, 820
zerovdw, 338
zlmfit, 160
zMatrix, 232
zmax, 79
zmin, 79