NAMD User's Guide

Version 2.7b1

M. Bhandarkar, A. Bhatele, E. Bohm, R. Brunner, F. Buelens, C. Chipot,
A. Dalke, S. Dixit, G. Fiorin, P. Freddolino, P. Grayson, J. Gullingsrud,
A. Gursoy, D. Hardy, C. Harrison, J. Hénin, W. Humphrey, D. Hurwitz,
N. Krawetz, S. Kumar, D. Kunzman, C. Lee, C. Mei, M. Nelson, J. Phillips,
O. Sarood, A. Shinozaki, G. Zheng, F. Zhu

March 23, 2009

Theoretical Biophysics Group
University of Illinois and Beckman Institute
405 N. Mathews
Urbana, IL 61801

Description

The NAMD *User's Guide* describes how to run and use the various features of the molecular dynamics program NAMD. This guide includes the capabilities of the program, how to use these capabilities, the necessary input files and formats, and how to run the program both on uniprocessor machines and in parallel.

NAMD Version 2.7b1

Authors: M. Bhandarkar, A. Bhatele, E. Bohm, R. Brunner, F. Buelens, C. Chipot,
A. Dalke, S. Dixit, G. Fiorin, P. Freddolino, P. Grayson, J. Gullingsrud, A. Gursoy,
D. Hardy, C. Harrison, J. Hénin, W. Humphrey, D. Hurwitz, N. Krawetz, S. Kumar,
D. Kunzman, C. Lee, C. Mei, M. Nelson, J. Phillips, O. Sarood, A. Shinozaki, G. Zheng,
F. Zhu

Theoretical Biophysics Group, Beckman Institute, University of Illinois.

NAMD Molecular Dynamics Software
Non-Exclusive, Non-Commercial Use License

Introduction

The University of Illinois at Urbana-Champaign has created its molecular dynamics software, NAMD, developed by the Theoretical Biophysics Group ("TBG") at Illinois' Beckman Institute available free of charge for non-commercial use by individuals, academic or research institutions and corporations for in-house business purposes only, upon completion and submission of the online registration form available from the NAMD web site `http://www.ks.uiuc.edu/Research/namd/`.

Commercial use of the NAMD software, or derivative works based thereon, REQUIRES A COMMERCIAL LICENSE. Commercial use includes: (1) integration of all or part of the Software into a product for sale, lease or license by or on behalf of Licensee to third parties, or (2) distribution of the Software to third parties that need it to commercialize product sold or licensed by or on behalf of Licensee. The University of Illinois will negotiate commercial-use licenses for NAMD upon request. These requests can be directed to namd@ks.uiuc.edu

Registration

Individuals may register in their own name or with their institutional or corporate affiliations. Registration information must include name, title, and e-mail of a person with signature authority to authorize and commit the individuals, academic or research institution, or corporation as necessary to the terms and conditions of the license agreement.

All parts of the information must be understood and agreed to as part of completing the form. Completion of the form is required before software access is granted. Pay particular attention to the authorized requester requirements above, and be sure that the form submission is authorized by the duly responsible person.

Registration will be administered by the NAMD development team.

UNIVERSITY OF ILLINOIS
NAMD MOLECULAR DYNAMICS SOFTWARE LICENSE AGREEMENT

Upon execution of this Agreement by the party identified below ("Licensee"), The Board of Trustees of the University of Illinois ("Illinois"), on behalf of The Theoretical Biophysics Group ("TBG") in the Beckman Institute, will provide the molecular dynamics software NAMD in Executable Code and/or Source Code form ("Software") to Licensee, subject to the following terms and conditions. For purposes of this Agreement, Executable Code is the compiled code, which is ready to run on Licensee's computer. Source code consists of a set of files which contain the actual program commands that are compiled to form the Executable Code.

1. The Software is intellectual property owned by Illinois, and all right, title and interest, including copyright, remain with Illinois. Illinois grants, and Licensee hereby accepts, a restricted, non-exclusive, non-transferable license to use the Software for academic, research and internal business purposes only e.g. not for commercial use (see Paragraph 7 below), without a fee. Licensee agrees to reproduce the copyright notice and other proprietary markings on all copies of the Software. Licensee has no right to transfer or sublicense the Software to any unauthorized person or entity. However, Licensee does have the right to make complimentary works that interoperate with NAMD, to freely distribute such complimentary works, and to direct others to the TBG server to obtain copies of NAMD itself.

2. Licensee may, at its own expense, modify the Software to make derivative works, for its own academic, research, and internal business purposes. Licensee's distribution of any derivative work is also subject to the same restrictions on distribution and use limitations that are specified herein for Illinois' Software. Prior to any such distribution the Licensee shall require the recipient of the Licensee's derivative work to first execute a license for NAMD with Illinois in accordance with the terms and conditions of this Agreement. Any derivative work should be clearly marked and renamed to notify users that it is a modified version and not the original NAMD code distributed by Illinois.

3. Except as expressly set forth in this Agreement, THIS SOFTWARE IS PROVIDED "AS IS" AND ILLINOIS MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY PATENT, TRADE-MARK, OR OTHER RIGHTS. LICENSEE ASSUMES THE ENTIRE RISK AS TO THE RE-SULTS AND PERFORMANCE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS. LICENSEE AGREES THAT UNIVERSITY SHALL NOT BE HELD LIABLE FOR ANY DI-RECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES WITH RESPECT TO ANY CLAIM BY LICENSEE OR ANY THIRD PARTY ON ACCOUNT OF OR ARISING FROM THIS AGREEMENT OR USE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS.

4. Licensee understands the Software is proprietary to Illinois. Licensee agrees to take all reasonable steps to insure that the Software is protected and secured from unauthorized disclosure, use, or release and will treat it with at least the same level of care as Licensee would use to protect and secure its own proprietary computer programs and/or information, but using no less than a reasonable standard of care. Licensee agrees to provide the Software only to any other person or entity who has registered with Illinois. If licensee is not registering as an individual but as an institution or corporation each member of the institution or corporation who has access to or uses Software must understand and agree to the terms of this license. If Licensee becomes aware of any unauthorized licensing, copying or use of the Software, Licensee shall promptly notify Illinois in

3

writing. Licensee expressly agrees to use the Software only in the manner and for the specific uses authorized in this Agreement.

5. By using or copying this Software, Licensee agrees to abide by the copyright law and all other applicable laws of the U.S. including, but not limited to, export control laws and the terms of this license. Illinois shall have the right to terminate this license immediately by written notice upon Licensee's breach of, or non-compliance with, any of its terms. Licensee may be held legally responsible for any copyright infringement that is caused or encouraged by its failure to abide by the terms of this license. Upon termination, Licensee agrees to destroy all copies of the Software in its possession and to verify such destruction in writing.

6. The user agrees that any reports or published results obtained with the Software will acknowledge its use by the appropriate citation as follows:

> NAMD was developed by the Theoretical Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign.

Any published work which utilizes NAMD shall include the following reference:

> James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26:1781-1802, 2005.

Electronic documents will include a direct link to the official NAMD page:

> `http://www.ks.uiuc.edu/Research/namd/`

One copy of each publication or report will be supplied to Illinois through Dr. Gila Budescu at the addresses listed below in Contact Information.

7. Should Licensee wish to make commercial use of the Software, Licensee will contact Illinois (namd@ks.uiuc.edu) to negotiate an appropriate license for such use. Commercial use includes: (1) integration of all or part of the Software into a product for sale, lease or license by or on behalf of Licensee to third parties, or (2) distribution of the Software to third parties that need it to commercialize product sold or licensed by or on behalf of Licensee.

8. Government Rights. Because substantial governmental funds have been used in the development of NAMD, any possession, use or sublicense of the Software by or to the United States government shall be subject to such required restrictions.

9. NAMD is being distributed as a research and teaching tool and as such, TBG encourages contributions from users of the code that might, at Illinois' sole discretion, be used or incorporated to make the basic operating framework of the Software a more stable, flexible, and/or useful product. Licensees that wish to contribute their code to become an internal portion of the Software may be required to sign an "Agreement Regarding Contributory Code for NAMD Software" before Illinois can accept it (contact namd@ks.uiuc.edu for a copy).

Contact Information

The best contact path for licensing issues is by e-mail to namd@ks.uiuc.edu or send correspondence to:

NAMD Team
Theoretical Biophysics Group
Beckman Institute
University of Illinois
405 North Mathews MC-251
Urbana, Illinois 61801 USA
FAX: (217) 244-6078

Contents

List of Figures

1 Introduction

NAMD is a parallel molecular dynamics program for UNIX platforms designed for high-performance simulations in structural biology. This document describes how to use NAMD, its features, and the platforms on which it runs. The document is divided into several sections:

Section 1 gives an overview of NAMD.

Section 2 lists the basics for getting started.

Section 3 describes NAMD file formats.

Section 4 explains PSF file generation with psfgen.

Section 5 presents the potential functions, non-bonded interactions, and full electrostatics.

Section 6 lists standard minimization and dynamics parameters.

Section 7 lists performance tuning parameters.

Section 8 explains user defined forces. conformation change calculations.

Section 9 describes collective variable-based calculations.

Section 10 explains alchemical free energy calculations.

Section 11 presents accelerated sampling methods.

Section 12 lists runtime analysis options.

Section 13 provides hints for X-PLOR users.

Section 14 provides sample configuration files.

Section 15 gives details on running NAMD.

Section 16 gives details on installing NAMD.

We have attempted to make this document complete and easy to understand and to make NAMD itself easy to install and run. We welcome your suggestions for improving the documentation or code at `namd@ks.uiuc.edu`.

1.1 NAMD and molecular dynamics simulations

Molecular dynamics (MD) simulations compute atomic trajectories by solving equations of motion numerically using empirical force fields, such as the CHARMM force field, that approximate the actual atomic force in biopolymer systems. Detailed information about MD simulations can be found in several books such as [1, 35]. In order to conduct MD simulations, various computer programs have been developed including X-PLOR [9] and CHARMM [8]. These programs were originally developed for serial machines. Simulation of large molecules, however, require enormous computing power. One way to achieve such simulations is to utilize parallel computers. In recent years, distributed memory parallel computers have been offering cost-effective computational power. NAMD was designed to run efficiently on such parallel machines for simulating large molecules.

NAMD is particularly well suited to the increasingly popular Beowulf-class PC clusters, which are quite similar to the workstation clusters for which is was originally designed. Future versions of NAMD will also make efficient use of clusters of multi-processor workstations or PCs.

NAMD has several important features:

- **Force Field Compatibility**
  The force field used by NAMD is the same as that used by the programs CHARMM [8] and X-PLOR [9]. This force field includes local interaction terms consisting of bonded interactions between 2, 3, and 4 atoms and pairwise interactions including electrostatic and van der Waals forces. This commonality allows simulations to migrate between these three programs.

- **Efficient Full Electrostatics Algorithms**
  NAMD incorporates the Particle Mesh Ewald (PME) algorithm, which takes the full electrostatic interactions into account. This algorithm reduces the computational complexity of electrostatic force evaluation from $O(N^2)$ to $O(N \log N)$.

- **Multiple Time Stepping**
  The velocity Verlet integration method [1] is used to advance the positions and velocities of the atoms in time. To further reduce the cost of the evaluation of long-range electrostatic forces, a multiple time step scheme is employed. The local interactions (bonded, van der Waals and electrostatic interactions within a specified distance) are calculated at each time step. The longer range interactions (electrostatic interactions beyond the specified distance) are only computed less often. This amortizes the cost of computing the electrostatic forces over several timesteps. A smooth splitting function is used to separate a quickly varying short-range portion of the electrostatic interaction from a more slowly varying long-range component. It is also possible to employ an intermediate timestep for the short-range non-bonded interactions, performing only bonded interactions every timestep.

- **Input and Output Compatibility**
  The input and output file formats used by NAMD are identical to those used by CHARMM and X-PLOR. Input formats include coordinate files in PDB format [4], structure files in X-PLOR PSF format, and energy parameter files in either CHARMM or X-PLOR formats. Output formats include PDB coordinate files and binary DCD trajectory files. These similarities assure that the molecular dynamics trajectories from NAMD can be read by CHARMM or X-PLOR and that the user can exploit the many analysis algorithms of the latter packages.

- **Dynamics Simulation Options**
  MD simulations may be carried out using several options, including

  - Constant energy dynamics,
  - Constant temperature dynamics via
    * Velocity rescaling,
    * Velocity reassignment,
    * Langevin dynamics,
  - Periodic boundary conditions,
  - Constant pressure dynamics via
    * Berendsen pressure coupling,

∗ Nosé-Hoover Langevin piston,

– Energy minimization,

– Fixed atoms,

– Rigid waters,

– Rigid bonds to hydrogen,

– Harmonic restraints,

– Spherical or cylindrical boundary restraints.

- **Easy to Modify and Extend**
  Another primary design objective for NAMD is extensibility and maintainability. In order to achieve this, it is designed in an object-oriented style with C++. Since molecular dynamics is a new field, new algorithms and techniques are continually being developed. NAMD's modular design allows one to integrate and test new algorithms easily. If you are contemplating a particular modification to NAMD you are encouraged to contact the developers at `namd@ks.uiuc.edu` for guidance.

- **Interactive MD simulations**
  A system undergoing simulation in NAMD may be viewed and altered with VMD; for instance, forces can be applied to a set of atoms to alter or rearrange part of the molecular structure. For more information on VMD, see `http://www.ks.uiuc.edu/Research/vmd/`.

- **Load Balancing**
  An important factor in parallel applications is the equal distribution of computational load among the processors. In parallel molecular simulation, a spatial decomposition that evenly distributes the computational load causes the region of space mapped to each processor to become very irregular, hard to compute and difficult to generalize to the evaluation of many different types of forces. NAMD addresses this problem by using a simple uniform spatial decomposition where the entire model is split into uniform cubes of space called *patches*. An initial load balancer assigns patches and the calculation of interactions among the atoms within them to processors such that the computational load is balanced as much as possible. During the simulation, an incremental load balancer monitors the load and performs necessary adjustments.

1.2   User feedback

If you have problems installing or running NAMD after reading this document, please send a complete description of the problem by email to `namd@ks.uiuc.edu`. If you discover and fix a problem not described in this manual we would appreciate if you would tell us about this as well, so we can alert other users and incorporate the fix into the public distribution.

We are interested in making NAMD more useful to the molecular modeling community. Your suggestions are welcome at `namd@ks.uiuc.edu`. We also appreciate hearing about how you are using NAMD in your work.

1.3   Acknowledgments

The authors would particularly like to thank the members of the Theoretical Biophysics Group, past and present, who have helped tremendously in making suggestions, pushing for new features, and testing bug-ridden code.

2 Getting Started

2.1 What is needed

Before running NAMD, explained in section 15, the following are be needed:

- A CHARMM force field in either CHARMM or X-PLOR format.

- An X-PLOR format PSF file describing the molecular structure.

- The initial coordinates of the molecular system in the form of a PDB file.

- A NAMD configuration file.

NAMD provides the `psfgen` utility, documented in Section 4, which is capable of generating the required PSF and PDB files by merging PDB files and guessing coordinates for missing atoms. If `psfgen` is insufficient for your system, we recommend that you obtain access to either CHARMM or X-PLOR, both of which are capable of generating the required files.

2.2 NAMD configuration file

Besides these input and output files, NAMD also uses a file referred to as the *configuration file*. This file specifies what dynamics options and values that NAMD should use, such as the number of timesteps to perform, initial temperature, etc. The options and values in this file control how the system will be simulated.

A NAMD configuration file contains a set of options and values. The options and values specified determine the exact behavior of NAMD, what features are active or inactive, how long the simulation should continue, etc. Section 2.2.1 describes how options are specified within a NAMD configuration file. Section 2.2.3 lists the parameters which are required to run a basic simulation. Section 13 describes the relation between specific NAMD and X-PLOR dynamics options. Several sample NAMD configuration files are shown in section 14.

2.2.1 Configuration parameter syntax

Each line in the configuration files consists of a *keyword* identifying the option being specified, and a *value* which is a parameter to be used for this option. The keyword and value can be separated by only white space:

```
keyword          value
```

or the keyword and value can be separated by an equal sign and white space:

```
keyword      =    value
```

Blank lines in the configuration file are ignored. Comments are prefaced by a `#` and may appear on the end of a line with actual values:

```
keyword          value          #  This is a comment
```

or may be at the beginning of a line:

```
#  This entire line is a comment . . .
```

Some keywords require several lines of data. These are generally implemented to either allow the data to be read from a file:

```
keyword             filename
```

or to be included inline using Tcl-style braces:

```
keyword {
  lots of data
}
```

The specification of the keywords is case insensitive so that any combination of upper and lower case letters will have the same meaning. Hence, `DCDfile` and `dcdfile` are equivalent. The capitalization in the values, however, may be important. Some values indicate file names, in which capitalization is critical. Other values such as `on` or `off` are case insensitive.

2.2.2   Tcl scripting interface and features

When compiled with Tcl (all released binaries) the config file is parsed by Tcl in a fully backwards compatible manner with the added bonus that any Tcl command may also be used. This alone allows:

- the "source" command to include other files (works w/o Tcl too!),

- the "print" command to display messages ("puts" is broken, sorry),

- environment variables through the env array ("$env(USER)"), and

- user-defined variables ("set base sim23", "dcdfile $base.dcd").

Additional features include:

- The "callback" command takes a 2-parameter Tcl procedure which is then called with a list of labels and a list of values during every timestep, allowing analysis, formatting, whatever.

- The "run" command takes a number of steps to run (overriding the now optional numsteps parameter, which defaults to 0) and can be called repeatedly. You can "run 0" just to get energies.

- The "minimize" command is similar to "run" and performs minimization for the specified number of force evaluations.

- The "output" command takes an output file basename and causes .coor, .vel, and .xsc files to be written with that name.

- Between "run" commands the reassignTemp, rescaleTemp, and langevinTemp parameters can be changed to allow simulated annealing protocols within a single config file. The useGroupPressure, useFlexibleCell, useConstantArea, useConstantRatio, LangevinPiston, LangevinPistonTarget, LangevinPistonPeriod, LangevinPistonDecay, LangevinPistonTemp, SurfaceTensionTarget, BerendsenPressure, BerendsenPressureTarget, BerendsenPressureCompressibility, and BerendsenPressureRelaxationTime parameters may be changed to

allow pressure equilibration. The fixedAtoms, constraintScaling, and nonbondedScaling parameters may be changed to preserve macromolecular conformation during minimization and equilibration (fixedAtoms may only be disabled, and requires that fixedAtomsForces is enabled to do this). The consForceScaling parameter may be changed to vary steering forces or to implement a time-varying electric field that affects specific atoms.

- The "checkpoint" and "revert" commands (no arguments) allow a scripted simulation to save and restore to a prior state.

- The "reinitvels" command reinitializes velocities to a random distribution based on the given temperature.

- The "rescalevels" command rescales velocities by the given factor.

- The "reloadCharges" command reads new atomic charges from the given file, which should contain one number for each atom, separated by spaces and/or line breaks.

- The "measure" command allows user-programmed calculations to be executed in order to facilitate automated methods. (For example, to revert or change a parameter.) A number of measure commands are included in the NAMD binary; the module has been designed to make it easy for users to add additional measure commands.

- The "coorfile" command allows NAMD to perform force and energy analysis on trajectory files. "coorfile open dcd `filename`" opens the specified DCD file for reading. "coorfile read" reads the next frame in the opened DCD file, replacing NAMD's atom coordinates with the coordinates in the frame, and returns 0 if successful or -1 if end-of-file was reached. "coorfile skip" skips past one frame in the DCD file; this is significantly faster than reading coordinates and throwing them away. "coorfile close" closes the file. The "coorfile" command is not available on the Cray T3E.

  Force and energy analysis are especially useful in the context of pair interaction calculations; see Sec. 12.1 for details, as well as the example scripts in Sec. 14.

Please note that while NAMD has traditionally allowed comments to be started by a # appearing anywhere on a line, Tcl only allows comments to appear where a new statement could begin. With Tcl config file parsing enabled (all shipped binaries) both NAMD and Tcl comments are allowed before the first "run" command. At this point only pure Tcl syntax is allowed. In addition, the ";#" idiom for Tcl comments will only work with Tcl enabled. NAMD has also traditionally allowed parameters to be specified as "param=value". This is supported, but only before the first "run" command. Some examples:

```
# this is my config file                        <- OK
reassignFreq 100 ; # how often to reset velocities  <- only w/ Tcl
reassignTemp 20 # temp to reset velocities to       <- OK before "run"
run 1000                                        <- now Tcl only
reassignTemp 40 ; # temp to reset velocities to     <- ";" is required
```

NAMD has also traditionally allowed parameters to be specified as "param=value" as well as "param value". This is supported, but only before the first "run" command. For an easy life, use "param value".

17

2.2.3 Required NAMD configuration parameters

The following parameters are *required* for every NAMD simulation:

- `numsteps` (page 57),

- `coordinates` (page 20),

- `structure` (page 20),

- `parameters` (page 20),

- `exclude` (page 45),

- `outputname` (page 21),

- one of the following three:

  - `temperature` (page 58),
  - `velocities` (page 20),
  - `binvelocities` (page 21).

These required parameters specify the most basic properties of the simulation. In addition, it is highly recommended that `pairlistdist` be specified with a value at least one greater than `cutoff`.

3 Input and Output Files

NAMD was developed to be compatible with existing molecular dynamics packages, especially the packages X-PLOR [9] and CHARMM [8]. To achieve this compatibility, the set of input files which NAMD uses to define a molecular system are identical to the input files used by X-PLOR and CHARMM. Thus it is trivial to move an existing simulation from X-PLOR or CHARMM to NAMD. A description of these molecular system definition files is given in Section 3.1.

In addition, the output file formats used by NAMD were chosen to be compatible with X-PLOR and CHARMM. In this way the output from NAMD can be analyzed using X-PLOR, CHARMM, or a variety of the other tools that have been developed for the existing output file formats. Descriptions of the output files formats are also given in Section 3.1.

3.1 File formats

3.1.1 PDB files

The PDB (Protein Data Bank) format is used to store coordinate or velocity data being input or output from NAMD. This is the standard format for coordinate data for most other molecular dynamics programs as well, including X-PLOR and CHARMM. A full description of this file format can be obtained from the PDB web site at `http://www.rcsb.org/pdb/`.

3.1.2 X-PLOR format PSF files

NAMD uses the same protein structure files that X-PLOR does. At this time, the easiest way to generate these files is using X-PLOR or CHARMM, although it is possible to build them by hand. CHARMM can generate an X-PLOR format PSF file with the command "`write psf card xplor`".

3.1.3 CHARMM19, CHARMM22, and CHARMM27 parameter files

NAMD supports CHARMM19, CHARMM22, and CHARMM27 parameter files in both X-PLOR and CHARMM formats. (X-PLOR format is the default, CHARMM format parameter files may be used given the parameter "`paraTypeCharmm on`".) For a full description of the format of commands used in these files, see the X-PLOR and CHARMM User's Manual [9].

3.1.4 DCD trajectory files

NAMD produces DCD trajectory files in the same format as X-PLOR and CHARMM. The DCD files are single precision binary FORTRAN files, so are transportable between computer architectures. They are not, unfortunately, transportable between big-endian (most workstations) and little endian (Intel) architectures. (This same caveat applies to binary velocity and coordinate files. The utility programs `flipdcd` and `flipbinpdb` are provided with the Linux/Intel version to reformat these files.) The exact format of these files is very ugly but supported by a wide range of analysis and display programs.

3.2 NAMD configuration parameters

3.2.1 Input files

- **coordinates** < coordinate PDB file >
  Acceptable Values: UNIX filename
  Description: The PDB file containing initial position coordinate data. Note that path names can be either absolute or relative. Only one value may be specified.

- **structure** < PSF file >
  Acceptable Values: UNIX filename
  Description: The X-PLOR format PSF file describing the molecular system to be simulated. Only one value may be specified.

- **parameters**
  Acceptable Values: UNIX filename
  Description: A CHARMM19, CHARMM22, or CHARMM27 parameter file that defines all or part of the parameters necessary for the molecular system to be simulated. At least one parameter file must be specified for each simulation. Multiple definitions (but only one file per definition) are allowed for systems that require more than one parameter file. The files will be read in the order that they appear in the configuration file. If duplicate parameters are read, a warning message is printed and the last parameter value read is used. Thus, the order that files are read can be important in cases where duplicate values appear in separate files.

- **paraTypeXplor** < Is the parameter file in X-PLOR format? >
  Acceptable Values: on or off
  Default Value: on
  Description: Specifies whether or not the parameter file(s) are in X-PLOR format. X-PLOR format is the default for parameter files! Caveat: The PSF file should be also constructed with X-PLOR in case of an X-PLOR parameter file because X-PLOR stores information about the multiplicity of dihedrals in the PSF file. See the X-PLOR manual for details.

- **paraTypeCharmm** < Is the parameter file in CHARMM format? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not the parameter file(s) are in CHARMM format. X-PLOR format is the default for parameter files! Caveat: The information about multiplicity of dihedrals will be obtained directly from the parameter file, and the full multiplicity will be used (same behavior as in CHARMM). If the PSF file originates from X-PLOR, consecutive multiple entries for the same dihedral (indicating the dihedral multiplicity for X-PLOR) will be ignored.

- **velocities** < velocity PDB file >
  Acceptable Values: UNIX filename
  Description: The PDB file containing the initial velocities for all atoms in the simulation. This is typically a restart file or final velocity file written by NAMD during a previous simulation. Either the **temperature** or the **velocities**/**binvelocities** option must be defined to determine an initial set of velocities. Both options cannot be used together.

- **binvelocities** < binary velocity file >
  Acceptable Values: UNIX filename
  Description: The binary file containing initial velocities for all atoms in the simulation. A binary velocity file is created as output from NAMD by activating the `binaryrestart` or `binaryoutput` options. The `binvelocities` option should be used as an alternative to `velocities`. Either the `temperature` or the `velocities`/`binvelocities` option must be defined to determine an initial set of velocities. Both options cannot be used together.

- **bincoordinates** < binary coordinate restart file >
  Acceptable Values: UNIX filename
  Description: The binary restart file containing initial position coordinate data. A binary coordinate restart file is created as output from NAMD by activating the `binaryrestart` or `binaryoutput` options. Note that, in the current implementation at least, the `bincoordinates` option must be used in addition to the `coordinates` option, but the positions specified by `coordinates` will then be ignored.

- **cwd** < default directory >
  Acceptable Values: UNIX directory name
  Description: The default directory for input and output files. If a value is given, all filenames that do not begin with a / are assumed to be in this directory. For example, if `cwd` is set to `/scr`, then a filename of `outfile` would be modified to `/scr/outfile` while a filename of `/tmp/outfile` would remain unchanged. If no value for `cwd` is specified, than all filenames are left unchanged *but are assumed to be relative to the directory which contains the configuration file given on the command line.*

3.2.2 Output files

- **outputname** < output PDB file >
  Acceptable Values: UNIX filename prefix
  Description: At the end of every simulation, NAMD writes two PDB files, one containing the final coordinates and another containing the final velocities of all atoms in the simulation. This option specifies the file prefix for these two files. The position coordinates will be saved to a file named as this prefix with `.coor` appended. The velocities will be saved to a file named as this prefix with `.vel` appended. For example, if the prefix specified using this option was `/tmp/output`, then the two files would be `/tmp/output.coor` and `/tmp/output.vel`.

- **binaryoutput** < use binary output files? >
  Acceptable Values: `yes` or `no`
  Default Value: `yes`
  Description: Activates the use of binary output files. If this option is set to `yes`, then the final output files will be written in binary rather than PDB format. Binary files preserve more accuracy between NAMD restarts than ASCII PDB files, but the binary files are not guaranteed to be transportable between computer architectures. (The utility program `flipbinpdb` is provided with the Linux/Intel version to reformat these files.)

- **restartname** < restart files >
  Acceptable Values: UNIX filename prefix
  Description: The prefix to use for restart filenames. NAMD produces PDB restart files

that store the current positions and velocities of all atoms at some step of the simulation. This option specifies the prefix to use for restart files in the same way that `outputname` specifies a filename prefix for the final positions and velocities. If `restartname` is defined then the parameter `restartfreq` must also be defined.

- `restartfreq` < frequency of restart file generation >
  Acceptable Values: positive integer
  Description: The number of timesteps between the generation of restart files. If `restartfreq` is defined, then `restartname` must also be defined.

- `restartsave` < use timestep in restart filenames? >
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: Appends the current timestep to the restart filename prefix, producing a sequence of restart files rather than only the last version written.

- `binaryrestart` < use binary restart files? >
  Acceptable Values: `yes` or `no`
  Default Value: `yes`
  Description: Activates the use of binary restart files. If this option is set to `yes`, then the restart files will be written in binary rather than PDB format. Binary files preserve more accuracy between NAMD restarts than ASCII PDB files, but the binary files are not guaranteed to be transportable between computer architectures. (The utility program `flipbinpdb` is provided with the Linux/Intel version to reformat these files.)

- `DCDfile` < coordinate trajectory output file >
  Acceptable Values: UNIX filename
  Description: The binary DCD position coordinate trajectory filename. This file stores the trajectory of all atom position coordinates using the same format (binary DCD) as X-PLOR. If `DCDfile` is defined, then `DCDfreq` must also be defined.

- `DCDfreq` < timesteps between writing coordinates to trajectory file >
  Acceptable Values: positive integer
  Description: The number of timesteps between the writing of position coordinates to the trajectory file. The initial positions will not be included in the trajectory file.

- `DCDUnitCell` < write unit cell data to dcd file? >
  Acceptable Values: `yes` or `no`
  Default Value: `yes` if periodic cell
  Description: If this option is set to `yes`, then DCD files will contain unit cell information in the style of Charmm DCD files. By default this option is enabled if the simulation cell is periodic in all three dimensions and disabled otherwise.

- `velDCDfile` < velocity trajectory output file >
  Acceptable Values: UNIX filename
  Description: The binary DCD velocity trajectory filename. This file stores the trajectory of all atom velocities using the same format (binary DCD) as X-PLOR. If `velDCDfile` is defined, then `velDCDfreq` must also be defined.

- **velDCDfreq** < timesteps between writing velocities to trajectory file >
  Acceptable Values: positive integer
  Description: The number of timesteps between the writing of velocities to the trajectory file. The initial velocities will not be included in the trajectory file.

3.2.3 Standard output

NAMD logs a variety of summary information to standard output. The standard units used by NAMD are Angstroms for length, kcal/mol for energy, Kelvin for temperature, and bar for pressure. Wallclock or CPU times are given in seconds unless otherwise noted.

BOUNDARY energy is from spherical boundary conditions and harmonic restraints, while MISC energy is from external electric fields and various steering forces. TOTAL is the sum of the various potential energies, and the KINETIC energy. TOTAL2 uses a slightly different kinetic energy that is better conserved during equilibration in a constant energy ensemble. TOTAL3 is another variation with much smaller short-time fluctuations that is also adjusted to have the same running average as TOTAL2. Defects in constant energy simulations are much easier to spot in TOTAL3 than in TOTAL or TOTAL2.

PRESSURE is the pressure calculated based on individual atoms, while GPRESSURE incorporates hydrogen atoms into the heavier atoms to which they are bonded, producing smaller fluctuations. The TEMPAVG, PRESSAVG, and GPRESSAVG are the average of temperature and pressure values since the previous ENERGY output; for the first step in the simulation they will be identical to TEMP, PRESSURE, and GPRESSURE.

- **outputEnergies** < timesteps between energy output >
  Acceptable Values: positive integer
  Default Value: 1
  Description: The number of timesteps between each energy output of NAMD. This value specifies how often NAMD should output the current energy values to **stdout** (which can be redirected to a file). By default, this is done every step. For long simulations, the amount of output generated by NAMD can be greatly reduced by outputting the energies only occasionally.

- **mergeCrossterms** < add crossterm energy to dihedral? >
  Acceptable Values: yes or no
  Default Value: yes
  Description: If crossterm (or CMAP) terms are present in the potential, the energy is added to the dihedral energy to avoid altering the energy output format. Disable this feature to add a separate "CROSS" field to the output.

- **outputMomenta** < timesteps between momentum output >
  Acceptable Values: nonnegative integer
  Default Value: 0
  Description: The number of timesteps between each momentum output of NAMD. If specified and nonzero, linear and angular momenta will be output to **stdout**.

- **outputPressure** < timesteps between pressure output >
  Acceptable Values: nonnegative integer
  Default Value: 0

Description: The number of timesteps between each pressure output of NAMD. If specified and nonzero, atomic and group pressure tensors will be output to **stdout**.

- **outputTiming** $<$ timesteps between timing output $>$
  Acceptable Values: nonnegative integer
  Default Value: the greater of `firstLdbStep` or $10\times$ `outputEnergies`
  Description: The number of timesteps between each timing output of NAMD. If nonzero, CPU and wallclock times and memory usage will be output to **stdout**. These data are from node 0 only; CPU times and memory usage for other nodes may vary.

3.3 AMBER force field parameters

AMBER format PARM file and coordinate file can be read by NAMD, which allows one to use AMBER force field to carry out all types of simulations that NAMD has supported. NAMD can read PARM files in either the format used in AMBER 6 or the new format defined in AMBER 7. The output of the simulation (restart file, DCD file, etc.) will still be in traditional format that has been used in NAMD.

- **amber** $<$ use AMBER format force field? $>$
  Acceptable Values: yes or no
  Default Value: no
  Description: If `amber` is set to on, then `parmfile` must be defined, and `structure` and `parameters` should not be defined.

- **parmfile** $<$ AMBER format PARM file $>$
  Acceptable Values: UNIX filename
  Description: This file contains complete topology and parameter information of the system.

- **ambercoor** $<$ AMBER format coordinate file $>$
  Acceptable Values: UNIX filename
  Description: This file contains the coordinates of all the atoms. Note that `coordinates` can also be used for PDB format coordinate file. When `amber` is set to on, either `ambercoor` or `coordinates` must be defined, but not both.

- **readexclusions** $<$ Read exclusions from PARM file? $>$
  Acceptable Values: yes or no
  Default Value: yes
  Description: PARM file explicitly gives complete exclusion (including 1-4 exclusions) information. When `readexclusions` is set to on, NAMD will read all exclusions from PARM file and will not add any more; alternatively, if `readexclusions` is set to off, NAMD will ignore the exclusions in PARM file and will automatically generate them according to the exclusion policy specified by `exclude`.

- **scnb** $<$ VDW 1-4 scaling factor $>$
  Acceptable Values: decimal ≥ 1.0
  Default Value: 2.0
  Description: Same meaning as SCNB in AMBER. Note that in NAMD, 1-4 vdw interactions are DIVIDED by `scnb`, whereas 1-4 electrostatic interactions are MULTIPLIED by `1-4scaling`. So `1-4scaling` should be set to the inverse of SCEE value used in AMBER.

Caveat:

1. Polarizable parameters in AMBER are not supported.

2. NAMD does not support the 10-12 potential terms in some old AMBER versions. When non-zero 10-12 parameter is encountered in PARM file, NAMD will terminate.

3. NAMD has several exclusion policy options, defined by `exclude`. The way AMBER dealing with exclusions corresponds to the "scaled1-4" in NAMD. So for simulations using AMBER force field, one would specify "exclude scaled1-4" in the configuration file, and set `1-4scaling` to the inverse value of SCEE as would be used in AMBER.

4. NAMD does not read periodic box lengths in PARM or coordinate file. They must be explicitly specified in NAMD configuration file.

5. By default, NAMD applies switching functions to the non-bond interactions within the cutoff distance, which helps to improve energy conservation, while AMBER does not use switching functions so it simply truncates the interactions at cutoff. However, if "authentic" AMBER cutoff simulations are desired, the switching functions could be turned off by specifying "switching off" in NAMD configuration file.

6. NAMD and AMBER may have different default values for some parameters (e.g., the tolerance of SHAKE). One should check other sections of this manual for accurate descriptions of the NAMD options.

Following are two examples of the NAMD configuration file to read AMBER force field and carry out simulation. They may help users to select proper NAMD options for AMBER force field. For the convenience of AMBER users, the AMBER 6 sander input files are given in the left for comparison, which would accomplish similar tasks in AMBER.

Example 1: Non-periodic boundary system, cutoff simulation

```
---AMBER----       ---NAMD---


 TITLE
 &cntrl
  ntb=0, igb=2,    # non-periodic, use cutoff for non-bond
  nstlim=1000,     numsteps       1000  # Num of total steps
  ntpr=50,         outputEnergies 50  # Energy output frequency
  ntwr=50,         restartfreq    50  # Restart file frequency
  ntwx=100,        DCDfreq        100  # Trajectory file frequency
  dt=0.001,        timestep       1  # in unit of fs (This is default)
  tempi=0.,        temperature    0  # Initial temp for velocity assignment
  cut=10.,         cutoff         10
                   switching      off  # Turn off the switching functions
  scee=1.2,        exclude        scaled1-4
                   1-4scaling     0.833333  # =1/1.2, default is 1.0
  scnb=2.0         scnb           2  # This is default
 &end

                   amber          on  # Specify this is AMBER force field
                   parmfile       prmtop  # Input PARM file
                   ambercoor      inpcrd  # Input coordinate file
                   outputname     md  # Prefix of output files
```

25

Example 2: Periodic boundary system, PME, NVE ensemble, using SHAKE algorithm

```
---AMBER----       ---NAMD---


 TITLE
 &cntrl
  ntc=2, ntf=2,    # SHAKE to the bond between each hydrogen and it mother atom
                   rigidBonds     all
  tol=0.0005,      rigidTolerance 0.0005  # Default is  0.00001
  nstlim=500,      numsteps       500  # Num of total steps
  ntpr=50,         outputEnergies 50  # Energy output frequency
  ntwr=100,        restartfreq    100  # Restart file frequency
  ntwx=100,        DCDfreq        100  # Trajectory file frequency
  dt=0.001,        timestep       1  # in unit of fs (This is default)
  tempi=300.,      temperature    300  # Initial temp for velocity assignment
  cut=9.,          cutoff         9
                   switching      off  # Turn off the switching functions
 &end
 &ewald           PME             on  # Use PME for electrostatic calculation
                  # Orthogonal periodic box size
  a=62.23,        cellBasisVector1   62.23  0  0
  b=62.23,        cellBasisVector2   0  62.23  0
  c=62.23,        cellBasisVector3   0  0  62.23
  nfft1=64,       PMEGridSizeX    64
  nfft2=64,       PMEGridSizeY    64
  nfft3=64,       PMEGridSizeZ    64
  ischrgd=1,      # NAMD doesn't force neutralization of charge
 &end
                  amber           on  # Specify this is AMBER force field
                  parmfile        FILENAME  # Input PARM file
                  ambercoor       FILENAME  # Input coordinate file
                  outputname      PREFIX  # Prefix of output files
                  exclude         scaled1-4
                  1-4scaling      0.833333  # =1/1.2, default is 1.0
```

3.4   GROMACS force field parameters

NAMD has the ability to load GROMACS ASCII topology (.top) and coordinate (.gro) files, which allows you to run most GROMACS simulations in NAMD. All simulation output will still be in the traditional NAMD formats.

- gromacs < use GROMACS format force field? >
  Acceptable Values: on or off
  Default Value: off
  Description: If gromacs is set to on, then grotopfile must be defined, and structure and parameters should not be defined.

- `grotopfile` < GROMACS format topology/parameter file >
  Acceptable Values: UNIX filename
  Description: This file contains complete topology and parameter information of the system.

- `grocoorfile` < GROMACS format coordinate file >
  Acceptable Values: UNIX filename
  Description: This file contains the coordinates of all the atoms. Note that `coordinates` can also be used for PDB format coordinate file. When `gromacs` is set to `on`, either `grocoorfile` or `coordinates` must be defined, but not both.

However, NAMD does not have support for many GROMACS-specific options:

- Dummies (fake atoms with positions generated from the positions of real atoms) are not supported.

- The GROMACS `pairs` section, where explicit 1–4 parameters are given between pairs of atoms, is not supported, since NAMD calculates its 1–4 interactions exclusively by type.

- Similarly, `exclusions` are not supported. The biggest problem here is that GROMACS RB dihedrals are supposed to imply exclusions, but NAMD does not support this.

- Constraints, restraints, and `settles` are not implemented in NAMD.

- In some cases, it may not work to override some but not all of the parameters for a bond, atom, etc. In this case, NAMD will generate an error and stop. The parser will sometimes not tolerate correct GROMACS files or fail to detect errors in badly formatted files.

- NAMD does not support all the types of bond potentials that exist in GROMACS, but approximates them with harmonic or sinusoidal potentials.

- NAMD does not read periodic box lengths in the coordinate file. They must be explicitly specified in the NAMD configuration file.

4 Creating PSF Structure Files

The `psfgen` structure building tool consists of a portable library of structure and file manipulation routines with a Tcl interface. Current capabilities include

- reading CHARMM topology files

- reading psf files in X-PLOR/NAMD format

- extracting sequence data from single segment PDB files

- generating a full molecular structure from sequence data

- applying patches to modify or link different segments

- writing NAMD and VMD compatible PSF structure files

- extracting coordinate data from PDB files

- constructing (guessing) missing atomic coordinates

- deleting selected atoms from the structure

- writing NAMD and VMD compatible PDB coordinate files

We are currently refining the interface of `psfgen` and adding features to create a complete molecular building solution. We welcome your feedback on this new tool.

4.1 Ordinary Usage

`psfgen` is currently distributed in two forms. One form is as a standalone program implemented as a Tcl interpreter which reads commands from standard output. You may use loops, variables, etc. as you would in a VMD or NAMD script. You may use psfgen interactively, but we expect it to be run most often with a script file redirected to standard input. The second form is as a Tcl package which can be imported into any Tcl application, including VMD. All the commands available to the standalone version of psfgen are available to the Tcl package; using `psfgen` within VMD lets you harness VMD's powerful atom selection capability, as well as instantly view the result of your structure building scripts. Examples of using `psfgen` both with and without VMD are provided in this document.

Generating PSF and PDB files for use with NAMD will typically consist of the following steps:

1. Preparing separate PDB files containing individual segments of protein, solvent, etc. before running `psfgen`.

2. Reading in the appropriate topology definition files and aliasing residue and atom names found in the PDB file to those found in the topology files. This will generally include selecting a default protonation state for histidine residues.

3. Generating the default structure using segment and pdb commands.

4. Applying additional patches to the structure.

5. Reading coordinates from the PDB files.

6. Deleting unwanted atoms, such as overlapping water molecules.

7. Guessing missing coordinates of hydrogens and other atoms.

8. Writing PSF and PDB files for use in NAMD.

4.1.1 Preparing separate PDB files

Many PDB files in the PDB databank contain multiple chains, corresponding to protein subunits, water, and other miscellaneous groups. Protein subunits are often identified by their chain ID in the PDB file. In psfgen, each of these groups must be assigned to their own *segment*. This applies most strictly in the case of protein chains, each of which must be assigned to its own segment so that N-terminal and C-terminal patches can be applied. You are free to group water molecules into whatever segments you choose.

Chains can be split up into their own PDB files using your favorite text editor and/or Unix shell commands, as illustrated in the BPTI example below. If you are using VMD you can also use atom selections to write pieces of the structure to separate files:

```
# Split a file containing protein and water into separate segments.
# Creates files named myfile_water.pdb, myfile_frag0.pdb, myfile_frag1.pdb,...
# Requires VMD.
mol load pdb myfile.pdb
set water [atomselect top water]
$water writepdb myfile_water.pdb
set protein [atomselect top protein]
set chains [lsort -unique [$protein get pfrag]]
foreach chain $chains {
  set sel [atomselect top "pfrag $chain"]
  $sel writepdb myfile_frag${chain}.pdb
}
```

4.1.2 Deleting unwanted atoms

The delatom command described below allows you to delete selected atoms from the structure. It's fine to remove atoms from your structure before building the PSF and PDB files, but you should never edit the PSF and PDB files created by psfgen by hand as it will probably mess up the internal numbering in the PSF file.

Very often the atoms you want to delete are water molecules that are either too far from the solute, or else outside of the periodic box you are trying to prepare. In either case VMD atom selections can be used to select the waters you want to delete. For example:

```
# Load a pdb and psf file into both psfgen and VMD.
resetpsf
readpsf myfile.psf
coordpdb myfile.pdb
mol load psf myfile.psf pdb myfile.pdb
# Select waters that are more than 10 Angstroms from the protein.
set badwater1 [atomselect top "name OH2 and not within 10 of protein"]
```

```
# Alternatively, select waters that are outside our periodic cell.
set badwater2 [atomselect top "name OH2 and (x<-30 or x>30 or y<-30 or>30
                                or z<-30 or z>30)"]
# Delete the residues corresponding to the atoms we selected.
foreach segid [$badwater1 get segid] resid [$badwater1 get resid] {
  delatom $segid $resid
}
# Have psfgen write out the new psf and pdb file (VMD's structure and
# coordinates are unmodified!).
writepsf myfile_chopwater.psf
writepdb myfile_chopwater.pdb
```

4.2 BPTI Example

To actually run this demo requires

- the program `psfgen` from any NAMD distribution,

- the CHARMM topology and parameter files `top_all22_prot.inp` and `par_all22_prot.inp` from `http://www.pharmacy.umaryland.edu/faculty/amackere/force_fields.htm`, and

- the BPTI PDB file `6PTI.pdb` available from the Protein Data Bank at `http://www.pdb.org/` by searching for `6PTI` and downloading the complete structure file in PDB format.

Building the BPTI structure

In this demo, we create the files `bpti.psf` and `bpti.pdb` in the output directory which can then be used for a simple NAMD simulation.

```
# File: bpti_example.tcl
# Requirements: topology file top_all22_prot.inp in directory toppar
#               PDB file 6PTI.pdb in current directory

# Create working directory; remove old output files
mkdir -p output
rm -f output/6PTI_protein.pdb output/6PTI_water.pdb

# (1) Split input PDB file into segments}
grep -v '^HETATM' 6PTI.pdb > output/6PTI_protein.pdb
grep 'HOH' 6PTI.pdb > output/6PTI_water.pdb

# (2) Embed the psfgen commands in this script
psfgen << ENDMOL

# (3) Read topology file
topology toppar/top_all22_prot.inp

# (4) Build protein segment
segment BPTI {
```

```
 pdb output/6PTI_protein.pdb
}

# (5) Patch protein segment
patch DISU BPTI:5 BPTI:55
patch DISU BPTI:14 BPTI:38
patch DISU BPTI:30 BPTI:51

# (6) Read protein coordinates from PDB file
pdbalias atom ILE CD1 CD     ; # formerly "alias atom ..."
coordpdb output/6PTI_protein.pdb BPTI

# (7) Build water segment
pdbalias residue HOH TIP3   ; # formerly "alias residue ..."
segment SOLV {
 auto none
 pdb output/6PTI_water.pdb
}

# (8) Read water coordinaes from PDB file
pdbalias atom HOH O OH2      ; # formerly "alias atom ..."
coordpdb output/6PTI_water.pdb SOLV

# (9) Guess missing coordinates
guesscoord

# (10) Write structure and coordinate files
writepsf output/bpti.psf
writepdb output/bpti.pdb

# End of psfgen commands
ENDMOL
```

Step-by-step explanation of the script:

(1) Split input PDB file into segments. 6PTI.pdb is the original file from the Protein Data Bank. It contains a single chain of protein and some PO4 and H2O HETATM records. Since each segment must have a separate input file, we remove all non-protein atom records using grep. If there were multiple chains we would have to split the file by hand. Create a second file containing only waters.

(2) Embed the psfgen commands in this script. Run the psfgen program, taking everything until "ENDMOL" as input. You may run psfgen interactively as well. Since psfgen is built on a Tcl interpreter, you may use loops, variables, etc., but you must use $$ for variables when inside a shell script. If you want, run psfgen and enter the following commands manually.

(3) Read topology file. Read in the topology definitions for the residues we will create. This must match the parameter file used for the simulation as well. Multiple topology files may be read in since psfgen and NAMD use atom type names rather than numbers in psf files.

(4) Build protein segment. Actually build a segment, calling it BPTI and reading the sequence of residues from the stripped pdb file created above. In addition to the pdb command, we could specify residues explicitly. Both angles and dihedrals are generated automatically unless "auto none" is added (which is required to build residues of water). The commands "first" and "last" may be used to change the default patches for the ends of the chain. The structure is built when the closing } is encountered, and some errors regarding the first and last residue are normal.

(5) Patch protein segment. Some patch residues (those not used to begin or end a chain) are applied after the segment is built. These contain all angle and dihedral terms explicitly since they were already generated. In this case we apply the patch for a disulfide link three separate times.

(6) Read protein coordinates from PDB file. The same file used to generate the sequence is now read to extract coordinates. In the residue ILE, the atom CD is called CD1 in the pdb file, so we use "pdbalias atom" to define the correct name. If the segment names in the pdb file match the name we gave in the segment statement, then we don't need to specify it again; in this case we do specify the segment, so that all atoms in the pdb file must belong to the segment.

(7) Build water segment. Build a segment for the crystal waters. The residue type for water depends on the model, so here we alias HOH to TIP3. Because CHARMM uses an additional H-H bond we must disable generation of angles and dihedrals for segments containing water. Then read the pdb file.

(8) Read water coordinates from PDB file. Alias the atom type for water oxygen as well and read coordinates from the file to the segment SOLV. Hydrogen doesn't show up in crystal structures so it is missing from this pdb file.

(9) Guessing missing coordinates. The tolopogy file contains default internal coordinates which can be used to guess the locations of many atoms, hydrogens in particular. In the output pdb file, the occupancy field of guessed atoms will be set to 0, atoms which are known are set to 1, and atoms which could not be guessed are set to -1. Some atoms are "poorly guessed" if needed bond lengths and angles were missing from the topology file. Similarly, waters with missing hydrogen coordinates are given a default orientation.

Write structure and coordinate files. Now that all of the atoms and bonds have been created, we can write out the psf structure file for the system. We also create the matching coordinate pdb file. The psf and pdb files are a matched set with identical atom ordering as needed by NAMD.

Using generated files in NAMD.

The files bpti.pdb and bpti.psf can now be used with NAMD, but the initial coordinates require minimization first. The following is an example NAMD configuration file for the BPTI example.

```
# NAMD configuration file for BPTI

# molecular system
structure output/bpti.psf

# force field
paratypecharmm on
parameters toppar/par_all22_prot.inp
exclude scaled1-4
1-4scaling 1.0

# approximations
switching on
switchdist 8
cutoff 12
pairlistdist 13.5
margin 0
stepspercycle 20

#integrator
timestep 1.0

#output
outputenergies 10
outputtiming 100
binaryoutput no

# molecular system
coordinates output/bpti.pdb

#output
outputname output/bpti
dcdfreq 1000

#protocol
temperature 0
reassignFreq 1000
reassignTemp 25
reassignIncr 25
reassignHold 300

#script

minimize 1000

run 20000
```

4.3 Building solvent around a protein

The following script illustrates how `psfgen` and VMD can be used together to add water around a protein structure. It assumes you already have a psf and pdb file for your protein, as well as a box of water which is large enough to contain the protein. For more information on how atomselections can be used within VMD scripts, see the VMD User's Guide.

```
proc addwater { psffile pdbfile watpsf watpdb } {
# Create psf/pdb files that contain both our protein as well as
# a box of equilibrated water.  The water box should be large enough
# to easily contain our protein.
resetpsf
readpsf $psffile
readpsf $watpsf
coordpdb $pdbfile
coordpdb $watpdb

# Load the combined structure into VMD
writepsf combine.psf
writepdb combine.pdb
mol load psf combine.psf pdb combine.pdb

# Assume that the segid of the water in watpsf is QQQ
# We want to delete waters outside of a box ten Angstroms
# bigger than the extent of the protein.
set protein [atomselect top "not segid QQQ"]
set minmax [measure minmax $protein]
foreach {min max} $minmax { break }
foreach {xmin ymin zmin} $min { break }
foreach {xmax ymax zmax} $max { break }
    set xmin [expr $xmin - 10]
    set ymin [expr $ymin - 10]
    set zmin [expr $zmin - 10]
    set xmax [expr $xmax + 10]
    set ymax [expr $ymax + 10]
    set zmax [expr $zmax + 10]

# Center the water on the protein.  Also update the coordinates held
# by psfgen.
set wat [atomselect top "segid QQQ"]
$wat moveby [vecsub [measure center $protein] [measure center $wat]]
foreach atom [$wat get {segid resid name x y z}] {
foreach {segid resid name x y z} $atom { break }
coord $segid $resid $name [list $x $y $z]
}

# Select waters that we don't want in the final structure.
```

```
set outsidebox [atomselect top "segid QQQ and (x <= $xmin or y <= $ymin \
or z <= $zmin or x >= $xmax or y >= $ymax or z >= $xmax)"]
set overlap [atomselect top "segid QQQ and within 2.4 of (not segid QQQ)"]

# Get a list of all the residues that are in the two selections, and delete
# those residues from the structure.
set reslist [concat [$outsidebox get resid] [$overlap get resid]]
set reslist [lsort -unique -integer $reslist]

foreach resid $reslist {
delatom QQQ $resid
}

# That should do it - write out the new psf and pdb file.
writepsf solvate.psf
writepdb solvate.pdb

# Delete the combined water/protein molecule and load the system that
# has excess water removed.
mol delete top
mol load psf solvate.psf pdb solvate.pdb

# Return the size of the water box
return [list [list $xmin $ymin $zmin] [list $xmax $ymax $zmax]]
}
```

4.4 List of Commands

- topology [list] *<file name>*
  Purpose: Read in molecular topology definitions from file.
  Arguments: *<file name>*: CHARMM format topology file.
  list: Lists all currently specified topology files.
  residues: Return a list of the known residue topologies.
  patches: Return a list of the known residue patches.
  Context: Beginning of script, before segment. May call multiple times.

- pdbalias residue *<alternate name>* *<real name>*
  Purpose: Provide translations from residues found in PDB files to proper residue names read
  in from topology definition files. Proper names from topology files will be used in generated
  PSF and PDB files. This command also exists under the deprecated name alias.
  Arguments: *<alternate name>*: Residue name found in PDB file.
  **: Residue name found in topology file.
  Context: Before reading sequence with pdb. May call multiple times.

- segment [segids] [resids] [residue] [first] [last] ** [*resid*] [*atom name*] {
  <commands> }
  Purpose: Build a segment of the molecule. A segment is typically a single chain of protein
  or DNA, with default patches applied to the termini. Segments may also contain pure solvent

or lipid. Options [segids] [resids] [residue] [first] [last] are used to query information about the specified segment.

Arguments: segids: Return a list of segids for the molecule in the current context.

resids: Return a list of resids for the molecule in the current context.

residue: Return the residue name of the residue in the given segment with the given resid.

atoms: Return a list of atoms for the given segment with the given resid.

coordinates: Return x, y, z coordinates for the given atom.

first: Returns the name of the patch that was applied to the beginning of the specified segment.

last: Returns the name of the patch that was applied to the end of the specified segment.

<*segment ID*>: Unique name for segment, 1–4 characters.

<*commands*>: Sequence of commands in Tcl syntax to build the primary structure of the segment, including auto, first, last, residue, pdb, etc.

Context: After topology definitions and residue aliases. May call multiple times. Structure information is generated at the end of every segment command.

- auto [angles] [dihedrals] [none]
  Purpose: Override default settings from topology file for automatic generation of angles and dihedrals for the current segment.
  Arguments: angles: Enable generation of angles from bonds.
  dihedrals: Enable generation of dihedrals from angles.
  none: Disable generation of angles and dihedrals.
  Context: Anywhere within segment, does not affect later segments.

- first <*patch name*>
  Purpose: Override default patch applied to first residue in segment. Default is read from topology file and may be residue-specific.
  Arguments: <*patch name*>: Single-target patch residue name or none.
  Context: Anywhere within segment, does not affect later segments.

- last <*patch name*>
  Purpose: Override default patch applied to last residue in segment. Default is read from topology file and may be residue-specific.
  Arguments: <*patch name*>: Single-target patch residue name or none.
  Context: Anywhere within segment, does not affect later segments.

- residue <*resid*> <*resname*> [*chain*]
  Purpose: Add a single residue to the end of the current segment.
  Arguments: <*resid*>: Unique name for residue, 1–5 characters, usually numeric. <*resname*>: Residue type name from topology file. <*chain*>: Single-character chain identifier.
  Context: Anywhere within segment.

- pdb <*file name*>
  Purpose: Extract sequence information from PDB file when building segment. Residue IDs will be preserved, residue names must match entries in the topology file or should be aliased before pdb is called.
  Arguments: <*file name*>: PDB file containing known or aliased residues.
  Context: Anywhere within segment.

- **mutate** *<resid>* *<resname>*
  Purpose: Change the type of a single residue in the current segment.
  Arguments: *<resid>*: Unique name for residue, 1–5 characters, usually numeric. *<resname>*: New residue type name from topology file.
  Context: Within segment, after target residue has been created.

- **patch** [list] *<patch residue name>* *<segid:resid>* [...]
  Purpose: Apply a patch to one or more residues. Patches make small modifications to the structure of residues such as converting one to a terminus, changing the protonation state, or creating disulphide bonds between a pair of residues.
  Arguments: list: Lists all patches applied explicitey using the command 'patch'.
  listall: Lists all currently applied patches including default patches.
  <patch residue name>: Name of patch residue from topology definition file.
  <segid:resid>: List of segment and residue pairs to which patch should be applied.
  Context: After one or more segments have been built.

- **regenerate** [angles] [dihedrals]
  Purpose: Remove all angles and/or dihedrals and completely regenerate them using the segment automatic generation algorithms. This is only needed if patches were applied that do not correct angles and bonds. Segment and file defaults are ignored, and angles/dihedrals for the entire molecule are regenerated from scratch.
  Arguments: angles: Enable generation of angles from bonds.
  dihedrals: Enable generation of dihedrals from angles.
  Context: After one or more segments have been built.

- **multiply** *<factor>* *<segid[:resid[:atomname]]>* [...]
  Purpose: Create multiple images of a set of atoms for use in locally enhanced sampling. The beta column of the output pdb file is set to 1...*<factor>* for each image. Multiple copies of bonds, angles, etc. are created. Atom, residue or segment names are not altered; images are distinguished only by beta value. This is not a normal molecular structure and may confuse other tools.
  Arguments: *<factor>*:
  <segid:resid:atomname>: segment, residue, or atom to be multiplied. If :resid is omitted the entire segment is multiplied; if :atomname is omitted the entire residue is multiplied. May be repeated as many times as necessary to include all atoms.
  Context: After one or more segments have been built, all patches applied, and coordinates guessed. The effects of this command may confuse other commands.

- **delatom** *<segid>* [resid] [atom name]
  Purpose: Delete one or more atoms. If only **segid** is specified, all atoms from that segment will be removed from the structure. If both **segid** and **resid** are specified, all atoms from just that residue will be removed. If **segid**, **resid**, and **atom name** are all specified, just a single atom will be removed.
  Arguments: *<segid>*: Name of segment.
  <resid>: Name of residue (optional).
  <atom name>: Name of atom (optional).
  Context: After all segments have been built and patched.

- `resetpsf`
  Purpose: Delete all segments in the structure. The topology definitions and aliases are left intact. If you want to clear the topology and aliases as well, use `psfcontext reset` instead.
  Arguments:
  Context: After one or more segments have been built.

- `psfcontext` [*context*] [`new`] [`delete`]
  Purpose: Switches between complete contexts, including structure, topology definitions, and aliases. If no arguments are provided, the current context is returned. If <*context*> or `new` is specified, a new context is entered and the old context is returned. If `delete` is also specified, the old context is destroyed and "deleted <*old context*>" is returned. An error is returned if the specified context does not exist or if `delete` was specified and the current context would still be in use. *It may be possible to write robust, error-tolerant code with this interface, but it would not be easy. Please employ the following revised* `psfcontext` *usage instead.*
  Arguments: <*context*>: Context ID returned by psfcontext.
  Context: At any time.

- `psfcontext reset`
  Purpose: Clears the structure, topology definitions, and aliases, creating clean environment just like a new context.
  Arguments:
  Context: At any time.

- `psfcontext create`
  Purpose: Creates a new context and returns its ID, but does not switch to it. This is different from `psfcontext new` above, which switches to the newly created context and returns the current context's ID.
  Arguments:
  Context: At any time.

- `psfcontext delete` <*context*>
  Purpose: Deletes the specified context. An error is returned if the specified context does not exist or would still be in use. This is different from `psfcontext` <***context***> `delete` above, which switches to the specified context and deletes the current one.
  Arguments: <*context*>: Context ID returned by psfcontext.
  Context: At any time.

- `psfcontext eval` <*context*> { <*commands*> }
  Purpose: Evaluates <*commands*> in the specified context, returning to the current context on exit. This should be totally robust, returning to the orignal context in case of errors and preventing its deletion when nested.
  Arguments: <*context*>: Context ID returned by `psfcontext create`.
  <*commands*>: Script to be executed in the specified context.
  Context: At any time.

- `psfcontext stats`
  Purpose: Returns the total numbers of contexts that have been created and destroyed. This is useful for checking if a script is leaking contexts.

Arguments:
Context: At any time.

- `writepsf` [charmm] [x-plor] [cmap] [nocmap] <*file name*>
  Purpose: Write out structure information as PSF file. A simplified session log is listed in the REMARKS section of the PSF file.
  Arguments: `charmm`: Use CHARMM format (numbers for atom types).
  `x-plor`: Use X-PLOR format (names for atom types), the default format required by NAMD.
  `cmap`: Write cross-term entries to PSF file if present, the default.
  `nocmap`: Do not write cross-term entries to PSF file, even if present.
  <*file name*>: PSF file to be generated.
  Context: After all segments have been built and patched.

- `readpsf` <*file name*>
  Purpose: Read in structure information from PSF file and adds it to the structure. It is an error if any segments in the PSF file already exist.
  Arguments: <*file name*>: PSF file in X-PLOR format (names for atom types).
  Context: Anywhere but within segment.

- `pdbalias atom` <*residue name*> <*alternate name*> <*real name*>
  Purpose: Provide translations from atom names found in PDB files to proper atom names read in from topology definition files. Proper names from topology files will be used in generated PSF and PDB files. This command also exists under the deprecated name `alias`.
  Arguments: <*residue name*>: Proper or aliased residue name.
  <*alternate name*>: Atom name found in PDB file.
  <*real name*>: Atom name found in topology file.
  Context: Before reading coordinates with coordpdb. May call multiple times.

- `coord` <*segid*> <*resid*> <*atomname*> <{ *x y z* }>
  Purpose: Set coordinates for a single atom.
  Arguments: <*segid*>: Segment ID of target atom.
  <*resid*>: Residue ID of target atom.
  <*atomname*>: Name of target atom.
  <{ *x y z* }>: Coordinates to be assigned.
  Context: After structure has been generated.

- `coordpdb` <*file name*> [*segid*]
  Purpose: Read coordinates from PDB file, matching segment, residue and atom names.
  Arguments: <*file name*>: PDB file containing known or aliased residues and atoms.
  <*segid*>: If specified override segment IDs in PDB file.
  Context: After segment has been generated and atom aliases defined.

- `guesscoord`
  Purpose: Guesses coordinates of atoms for which they were not explicitly set. Calculation is based on internal coordinate hints contained in topology definition files. When these are insufficient, wild guesses are attempted based on bond lengths of 1 Å and angles of 109°.
  Arguments: None.
  Context: After stucture has been generated and known coordinates read in.

- writepdb <*file name*>

  Purpose: Writes PDB file containing coordinates. Atoms order is identical to PSF file generated by writepsf (unless structure has been changed). The O field is set to 1 for atoms with known coordinates, 0 for atoms with guessed coordinates, and -1 for atoms with no coordinate data available (coordinates are set to 0 for these atoms).

  Arguments: <*file name*>: PDB file to be written.

  Context: After structure and coordinates are complete.

4.5   Example of a Session Log

The command "writepsf" prints a simple session log as "REMARKS" at the beginning of the PSF file. The log contains information about applied patches and used topology files which not stored in the standard records of PSF files. These informations are also available after a PSF file was read by command "readpsf". Here'a a simple axample:

```
PSF


       1 !NTITLE
 REMARKS original generated structure x-plor psf file
 REMARKS 4 patches were applied to the molecule.
 REMARKS topology 1LOV_autopsf-temp.top
 REMARKS segment P1 { first NTER; last CTER; auto angles dihedrals }
 REMARKS segment O1 { first NONE; last NONE; auto none  }
 REMARKS segment W1 { first NONE; last NONE; auto none  }
 REMARKS defaultpatch NTER P1:1
 REMARKS defaultpatch CTER P1:104
 REMARKS patch DISU P1:10  P1:2
 REMARKS patch DISU P1:103  P1:6


    1704 !NATOM
       1 P1   1    ALA  N    NH3   -0.300000        14.0070           0
...
```

All patches that were applied explicitely using the "patch" command are listed following the keyword "patch", but the patches that result from default patching like the first and last patches of a segment are marked as "defaultpatch". Further the segment based patching rules are listed along with the angle/dihedral autogeneration rules.

5 Force Field Parameters

5.1 Potential energy functions

Evaluating the force is the most computationally demanding part of molecular dynamics. The force is the negative gradient of a scalar potential energy function,

$$\vec{F}(\vec{r}) = -\nabla U(\vec{r}),\tag{1}$$

and, for systems of biomolecules, this potential function involves the summing,

$$U(\vec{r}) = \sum U_{\text{bonded}}(\vec{r}) + \sum U_{\text{nonbonded}}(\vec{r}),\tag{2}$$

over a large number of bonded and nonbonded terms. The bonded potential terms involve 2–, 3–, and 4–body interactions of covalently bonded atoms, with $O(N)$ terms in the summation. The nonbonded potential terms involve interactions between all pairs of atoms (usually excluding pairs of atoms already involved in a bonded term), with $O(N^2)$ terms in the summation, although fast evaluation techniques are used to compute good approximations to their contribution to the potential with $O(N)$ or $O(N \log N)$ computational cost.

5.1.1 Bonded potential energy terms

The bonded potential terms involve 2–, 3–, and 4–body interactions of covalently bonded atoms.

The 2–body spring bond potential describes the harmonic vibrational motion between an (i, j)–pair of covalently bonded atoms,

$$U_{\text{bond}} = k(r_{ij} - r_0)^2,\tag{3}$$

where $r_{ij} = \|\vec{r}_j - \vec{r}_i\|$ gives the distance between the atoms, r_0 is the equilibrium distance, and k is the spring constant.

The 3–body angular bond potential describes the angular vibrational motion occurring between an (i, j, k)–triple of covalently bonded atoms,

$$U_{\text{angle}} = k_\theta (\theta - \theta_0)^2 + k_{\text{ub}}(r_{ik} - r_{\text{ub}})^2,\tag{4}$$

where, in the first term, θ is the angle in radians between vectors $\vec{r}_{ij} = \vec{r}_j - \vec{r}_i$ and $\vec{r}_{kj} = \vec{r}_j - \vec{r}_k$, θ_0 is the equilibrium angle, and k_θ is the angle constant. The second term is the Urey–Bradley term used to describe a (noncovalent) spring between the outer i and k atoms, active when constant $k_{\text{ub}} \neq 0$, where, like the spring bond, $r_{ik} = \|\vec{r}_k - \vec{r}_i\|$ gives the distance between the pair of atoms and r_{ub} is the equilibrium distance.

The 4–body torsion angle (also known as dihedral angle) potential describes the angular spring between the planes formed by the first three and last three atoms of a consecutively bonded (i, j, k, l)–quadruple of atoms,

$$U_{\text{tors}} = \begin{cases} k(1 + \cos(n\psi + \phi)) & \text{if } n > 0, \\ k(\psi - \phi)^2 & \text{if } n = 0, \end{cases}\tag{5}$$

where ψ is the angle in radians between the (i, j, k)–plane and the (j, k, l)–plane. The integer constant n is nonnegative and indicates the periodicity. For $n > 0$, ϕ is the phase shift angle and k is the multiplicative constant. For $n = 0$, ϕ acts as an equilibrium angle and the units of k change to potential/rad^2. A given (i, j, k, l)–quadruple of atoms might contribute multiple terms to the potential, each with its own parameterization. The use of multiple terms for a torsion angle allows for complex angular variation of the potential, effectively a truncated Fourier series.

5.1.2 Nonbonded potential energy terms

The nonbonded potential terms involve interactions between all (i, j)–pairs of atoms, usually excluding pairs of atoms already involved in a bonded term. Even using a fast evaluation methods the cost of computing the nonbonded potentials dominates the work required for each time step of an MD simulation.

The Lennard–Jones potential accounts for the weak dipole attraction between distant atoms and the hard core repulsion as atoms become close,

$$U_{\text{LJ}} = (-E_{\text{min}}) \left[\left(\frac{R_{\text{min}}}{r_{ij}} \right)^{12} - 2 \left(\frac{R_{\text{min}}}{r_{ij}} \right)^{6} \right], \tag{6}$$

where $r_{ij} = \|\vec{r}_j - \vec{r}_i\|$ gives the distance between the pair of atoms. The parameter $E_{\text{min}} = U_{\text{LJ}}(R_{\text{min}})$ is the minimum of the potential term ($E_{\text{min}} < 0$, which means that $-E_{\text{min}}$ is the well-depth). The Lennard–Jones potential approaches 0 rapidly as r_{ij} increases, so it is usually truncated (smoothly shifted) to 0 past a cutoff radius, requiring $O(N)$ computational cost.

The electrostatic potential is repulsive for atomic charges with the same sign and attractive for atomic charges with opposite signs,

$$U_{\text{elec}} = \epsilon_{14} \frac{C q_i q_j}{\epsilon_0 r_{ij}}, \tag{7}$$

where $r_{ij} = \|\vec{r}_j - \vec{r}_i\|$ gives the distance between the pair of atoms, and q_i and q_j are the charges on the respective atoms. Coulomb's constant C and the dielectric constant ϵ_0 are fixed for all electrostatic interactions. The parameter ϵ_{14} is a unitless scaling factor whose value is 1, except for a modified 1–4 interaction, where the pair of atoms is separated by a sequence of three covalent bonds (so that the atoms might also be involved in a torsion angle interaction), in which case $\epsilon_{14} = \varepsilon$, for a fixed constant $0 \leq \varepsilon \leq 1$. Although the electrostatic potential may be computed with a cutoff like the Lennard–Jones potential, the $1/r$ potential approaches 0 much more slowly than the $1/r^6$ potential, so neglecting the long range electrostatic terms can degrade qualitative results, especially for highly charged systems. There are other fast evaluation methods that approximate the contribution to the long range electrostatic terms that require $O(N)$ or $O(N \log N)$ computational cost, depending on the method.

5.2 Non-bonded interactions

NAMD has a number of options that control the way that non-bonded interactions are calculated. These options are interrelated and can be quite confusing, so this section attempts to explain the behavior of the non-bonded interactions and how to use these parameters.

5.2.1 Van der Waals interactions

The simplest non-bonded interaction is the van der Waals interaction. In NAMD, van der Waals interactions are always truncated at the cutoff distance, specified by `cutoff`. The main option that effects van der Waals interactions is the `switching` parameter. With this option set to `on`, a smooth switching function will be used to truncate the van der Waals potential energy smoothly at the cutoff distance. A graph of the van der Waals potential with this switching function is shown in Figure 1. If `switching` is set to `off`, the van der Waals energy is just abruptly truncated at the cutoff distance, so that energy may not be conserved.
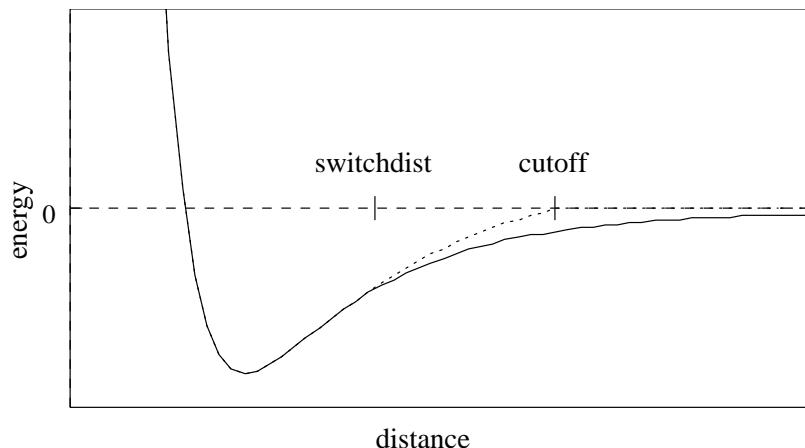
Figure 1: Graph of van der Waals potential with and without the application of the switching function. With the switching function active, the potential is smoothly reduced to 0 at the cutoff distance. Without the switching function, there is a discontinuity where the potential is truncated.

The switching function used is based on the X-PLOR switching function. The parameter `switchdist` specifies the distance at which the switching function should start taking effect to bring the van der Waals potential to 0 smoothly at the cutoff distance. Thus, the value of `switchdist` must always be less than that of `cutoff`.

5.2.2 Electrostatic interactions

The handling of electrostatics is slightly more complicated due to the incorporation of multiple timestepping for full electrostatic interactions. There are two cases to consider, one where full electrostatics is employed and the other where electrostatics are truncated at a given distance.

First let us consider the latter case, where electrostatics are truncated at the cutoff distance. Using this scheme, all electrostatic interactions beyond a specified distance are ignored, or assumed to be zero. If `switching` is set to `on`, rather than having a discontinuity in the potential at the cutoff distance, a shifting function is applied to the electrostatic potential as shown in Figure 2. As this figure shows, the shifting function shifts the entire potential curve so that the curve intersects the x-axis at the cutoff distance. This shifting function is based on the shifting function used by X-PLOR.

Next, consider the case where full electrostatics are calculated. In this case, the electrostatic interactions are not truncated at any distance. In this scheme, the `cutoff` parameter has a slightly different meaning for the electrostatic interactions — it represents the *local interaction distance*, or distance within which electrostatic pairs will be directly calculated every timestep. Outside of this distance, interactions will be calculated only periodically. These forces will be applied using a multiple timestep integration scheme as described in Section 6.3.4.

5.2.3 Non-bonded force field parameters

- `cutoff` < local interaction distance common to both electrostatic and van der Waals calculations (Å) >
  Acceptable Values: positive decimal
  Description: See Section 5.2 for more information.
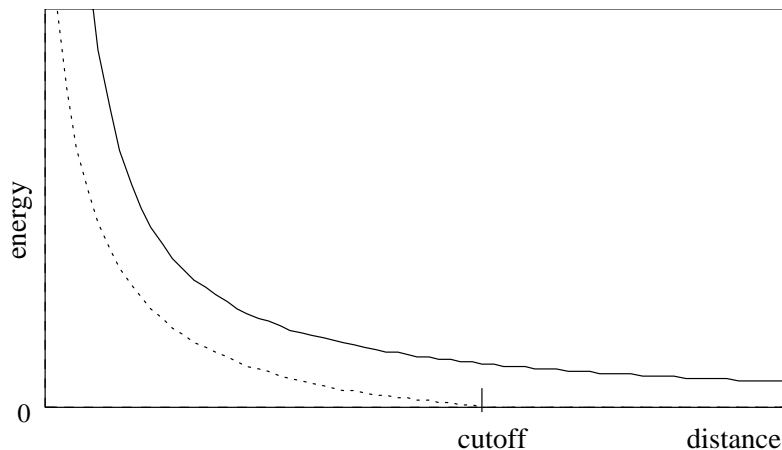
Figure 2: Graph showing an electrostatic potential with and without the application of the shifting function.
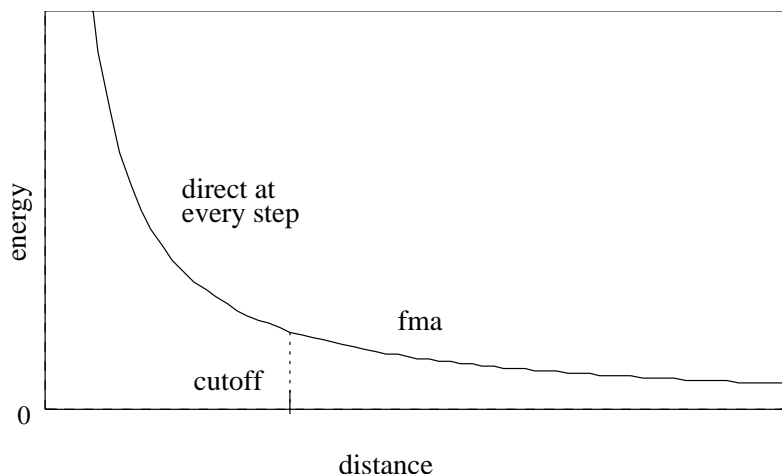
Figure 3: Graph showing an electrostatic potential when full electrostatics are used within NAMD, with one curve portion calculated directly and the other calculated using DPMTA.

- **switching** < use switching function? >
  Acceptable Values: on or off
  Default Value: on
  Description: If switching is specified to be off, then a truncated cutoff is performed. If switching is turned on, then smoothing functions are applied to both the electrostatics and van der Waals forces. For a complete description of the non-bonded force parameters see Section 5.2. If switching is set to on, then switchdist must also be defined.

- **switchdist** < distance at which to activate switching/splitting function for electrostatic and van der Waals calculations (Å) >
  Acceptable Values: positive decimal $\leq$ cutoff
  Description: Distance at which the switching function should begin to take effect. This parameter only has meaning if switching is set to on. The value of switchdist must be less than or equal to the value of cutoff, since the switching function is only applied on the range from switchdist to cutoff. For a complete description of the non-bonded force parameters

44

see Section 5.2.

- **exclude** $<$ exclusion policy to use $>$
  Acceptable Values: `none`, `1-2`, `1-3`, `1-4`, or `scaled1-4`
  Description: This parameter specifies which pairs of bonded atoms should be excluded from non-bonded interactions. With the value of `none`, no bonded pairs of atoms will be excluded. With the value of `1-2`, all atom pairs that are directly connected via a linear bond will be excluded. With the value of `1-3`, all `1-2` pairs will be excluded along with all pairs of atoms that are bonded to a common third atom (i.e., if atom A is bonded to atom B and atom B is bonded to atom C, then the atom pair A-C would be excluded). With the value of `1-4`, all `1-3` pairs will be excluded along with all pairs connected by a set of two bonds (i.e., if atom A is bonded to atom B, and atom B is bonded to atom C, and atom C is bonded to atom D, then the atom pair A-D would be excluded). With the value of `scaled1-4`, all `1-3` pairs are excluded and all pairs that match the `1-4` criteria are modified. The electrostatic interactions for such pairs are modified by the constant factor defined by `1-4scaling`. The van der Waals interactions are modified by using the special 1-4 parameters defined in the parameter files.

- **dielectric** $<$ dielectric constant for system $>$
  Acceptable Values: decimal ≥ 1.0
  Default Value: 1.0
  Description: Dielectric constant for the system. A value of 1.0 implies no modification of the electrostatic interactions. Any larger value will lessen the electrostatic forces acting in the system.

- **nonbondedScaling** $<$ scaling factor for nonbonded forces $>$
  Acceptable Values: decimal ≥ 0.0
  Default Value: 1.0
  Description: Scaling factor for electrostatic and van der Waals forces. A value of 1.0 implies no modification of the interactions. Any smaller value will lessen the nonbonded forces acting in the system.

- **1-4scaling** $<$ scaling factor for 1-4 interactions $>$
  Acceptable Values: $0 \leq$ decimal ≤ 1
  Default Value: 1.0
  Description: Scaling factor for 1-4 interactions. This factor is only used when the `exclude` parameter is set to `scaled1-4`. In this case, this factor is used to modify the electrostatic interactions between 1-4 atom pairs. If the `exclude` parameter is set to anything but `scaled1-4`, this parameter has no effect regardless of its value.

- **vdwGeometricSigma** $<$ use geometric mean to combine L-J sigmas $>$
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: Use geometric mean, as required by OPLS, rather than traditional arithmetic mean when combining Lennard-Jones sigma parameters for different atom types.

- **limitdist** $<$ maximum distance between pairs for limiting interaction strength(Å) $>$
  Acceptable Values: non-negative decimal
  Default Value: 0.

Description: The electrostatic and van der Waals potential functions diverge as the distance between two atoms approaches zero. The potential for atoms closer than `limitdist` is instead treated as $ar^2 + c$ with parameters chosen to match the force and potential at `limitdist`. This option should primarily be useful for alchemical free energy perturbation calculations, since it makes the process of creating and destroying atoms far less drastic energetically. The larger the value of `limitdist` the more the maximum force between atoms will be reduced. In order to not alter the other interactions in the simulation, `limitdist` should be less than the closest approach of any non-bonded pair of atoms; 1.3 Å appears to satisfy this for typical simulations but the user is encouraged to experiment. There should be no performance impact from enabling this feature.

5.2.4   PME parameters

PME stands for Particle Mesh Ewald and is an efficient full electrostatics method for use with periodic boundary conditions. None of the parameters should affect energy conservation, although they may affect the accuracy of the results and momentum conservation.

- **PME** < Use particle mesh Ewald for electrostatics? >
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: Turns on particle mesh Ewald.

- **PMETolerance** < PME direct space tolerance >
  Acceptable Values: positive decimal
  Default Value: 10^{-6}
  Description: Affects the value of the Ewald coefficient and the overall accuracy of the results.

- **PMEInterpOrder** < PME interpolation order >
  Acceptable Values: positive integer
  Default Value: 4 (cubic)
  Description: Charges are interpolated onto the grid and forces are interpolated off using this many points, equal to the order of the interpolation function plus one.

- **PMEGridSpacing** < maximum space between grid points >
  Acceptable Values: positive real
  Description: The grid spacing partially determines the accuracy and efficiency of PME. If any of the grid sizes below are not set, then PMEGridSpacing must be set (recommended value is 1.0 Å) and will be used to calculate them. If a grid size is set, then the grid spacing must be at least PMEGridSpacing (if set, or a very large default of 1.5).

- **PMEGridSizeX** < number of grid points in x dimension >
  Acceptable Values: positive integer
  Description: The grid size partially determines the accuracy and efficiency of PME. For speed, `PMEGridSizeX` should have only small integer factors (2, 3 and 5).

- **PMEGridSizeY** < number of grid points in y dimension >
  Acceptable Values: positive integer
  Description: The grid size partially determines the accuracy and efficiency of PME. For speed, `PMEGridSizeY` should have only small integer factors (2, 3 and 5).

- **PMEGridSizeZ** $<$ number of grid points in z dimension $>$
  Acceptable Values: positive integer
  Description: The grid size partially determines the accuracy and efficiency of PME. For speed, `PMEGridSizeZ` should have only small integer factors (2, 3 and 5).

- **PMEProcessors** $<$ processors for FFT and reciprocal sum $>$
  Acceptable Values: positive integer
  Default Value: larger of x and y grid sizes up to all available processors
  Description: For best performance on some systems and machines, it may be necessary to restrict the amount of parallelism used. Experiment with this parameter if your parallel performance is poor when PME is used.

- **FFTWEstimate** $<$ Use estimates to optimize FFT? $>$
  Acceptable Values: yes or no
  Default Value: no
  Description: Do not optimize FFT based on measurements, but on FFTW rules of thumb. This reduces startup time, but may affect performance.

- **FFTWUseWisdom** $<$ Use FFTW wisdom archive file? $>$
  Acceptable Values: yes or no
  Default Value: yes
  Description: Try to reduce startup time when possible by reading FFTW "wisdom" from a file, and saving wisdom generated by performance measurements to the same file for future use. This will reduce startup time when running the same size PME grid on the same number of processors as a previous run using the same file.

- **FFTWWisdomFile** $<$ name of file for FFTW wisdom archive $>$
  Acceptable Values: file name
  Default Value: FFTW_NAMD_*version_platform*.txt
  Description: File where FFTW wisdom is read and saved. If you only run on one platform this may be useful to reduce startup times for all runs. The default is likely sufficient, as it is version and platform specific.

5.2.5 Full direct parameters

The direct computation of electrostatics is not intended to be used during real calculations, but rather as a testing or comparison measure. Because of the $\mathcal{O}(N^2)$ computational complexity for performing direct calculations, this is *much* slower than using DPMTA or PME to compute full electrostatics for large systems. In the case of periodic boundary conditions, the nearest image convention is used rather than a full Ewald sum.

- **FullDirect** $<$ calculate full electrostatics directly? $>$
  Acceptable Values: yes or no
  Default Value: no
  Description: Specifies whether or not direct computation of full electrostatics should be performed.

5.2.6 DPME parameters

DPME is an implementation of PME that is no longer included in the released NAMD binaries. We recommend that you use the current PME implementation.

- **useDPME** < Use old DPME code? >
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: Switches to old DPME implementation of particle mesh Ewald. The new code is faster and allows non-orthogonal cells so you probably just want to leave this option turned off. If you set `cellOrigin` to something other than $(0, 0, 0)$ the energy may differ slightly between the old and new implementations. *DPME is no longer included in released binaries.*

5.2.7 DPMTA parameters

DPMTA is no longer included in the released NAMD binaries. We recommend that you instead use PME with a periodic system because it conserves energy better, is more efficient, and is better parallelized. If you must have the fast multipole algorithm you may compile NAMD yourself.

These parameters control the options to DPMTA, an algorithm used to provide full electrostatic interactions. DPMTA is a modified version of the FMA (Fast Multipole Algorithm) and, unfortunately, most of the parameters still refer to FMA rather than DPMTA for historical reasons. Don't be confused!

For a further description of how exactly full electrostatics are incorporated into NAMD, see Section 6.3.4. For a greater level of detail about DPMTA and the specific meaning of its options, see the DPMTA distribution which is available via anonymous FTP from the site `ftp.ee.duke.edu` in the directory `/pub/SciComp/src`.

- **FMA** < use full electrostatics? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether or not the DPMTA algorithm from Duke University should be used to compute the full electrostatic interactions. If set to `on`, DPMTA will be used with a multiple timestep integration scheme to provide full electrostatic interactions as detailed in Section 6.3.4. *DPMTA is no longer included in released binaries.*

- **FMALevels** < number of levels to use in multipole expansion >
  Acceptable Values: positive integer
  Default Value: 5
  Description: Number of levels to use for the multipole expansion. This parameter is only used if `FMA` is set to `on`. A value of 4 should be sufficient for systems with less than 10,000 atoms. A value of 5 or greater should be used for larger systems.

- **FMAMp** < number of multipole terms to use for FMA >
  Acceptable Values: positive integer
  Default Value: 8
  Description: Number of terms to use in the multipole expansion. This parameter is only used if `FMA` is set to `on`. If the `FMAFFT` is set to `on`, then this value must be a multiple of 4. The default value of 8 should be suitable for most applications.

- `FMAFFT` < use DPMTA FFT enhancement? >
  Acceptable Values: `on` or `off`
  Default Value: `on`
  Description: Specifies whether or not the DPMTA code should use the FFT enhancement feature. This parameter is only used if `FMA` is set to `on`. If `FMAFFT` is set to `on`, the value of `FMAMp` must be set to a multiple of 4. This feature offers substantial benefits only for values of `FMAMp` of 8 or greater. This feature will substantially increase the amount of memory used by DPMTA.

- `FMAtheta` < DPMTA theta parameter (radians) >
  Acceptable Values: decimal
  Default Value: 0.715
  Description: This parameter specifies the value of the theta parameter used in the DPMTA calculation. The default value is based on recommendations by the developers of the code.

- `FMAFFTBlock` < blocking factor for FMA FFT >
  Acceptable Values: positive integer
  Default Value: 4
  Description: The blocking factor for the FFT enhancement to DPMTA. This parameter is only used if both `FMA` and `FMAFFT` are set to `on`. The default value of 4 should be suitable for most applications.

5.3   Constraints and Restraints

5.3.1   Bond constraint parameters

- `rigidBonds` < controls if and how ShakeH is used >
  Acceptable Values: `none`, `water`, `all`
  Default Value: `none`
  Description: When `water` is selected, the hydrogen-oxygen and hydrogen-hydrogen distances in waters are constrained to the nominal length or angle given in the parameter file, making the molecules completely rigid. When `rigidBonds` is `all`, waters are made rigid as described above and the bond between each hydrogen and the (one) atom to which it is bonded is similarly constrained. For the default case `none`, no lengths are constrained.

- `rigidTolerance` < allowable bond-length error for ShakeH (Å) >
  Acceptable Values: positive decimal
  Default Value: 1.0e-8
  Description: The ShakeH algorithm is assumed to have converged when all constrained bonds differ from the nominal bond length by less than this amount.

- `rigidIterations` < maximum ShakeH iterations >
  Acceptable Values: positive integer
  Default Value: 100
  Description: The maximum number of iterations ShakeH will perform before giving up on constraining the bond lengths. If the bond lengths do not converge, a warning message is printed, and the atoms are left at the final value achieved by ShakeH. Although the default value is 100, convergence is usually reached after fewer than 10 iterations.

- **rigidDieOnError** < maximum ShakeH iterations >
  Acceptable Values: on or off
  Default Value: on
  Description: Exit and report an error if rigidTolerance is not achieved after rigidIterations.

- **useSettle** < Use SETTLE for waters. >
  Acceptable Values: on or off
  Default Value: on
  Description: If rigidBonds are enabled then use the non-iterative SETTLE algorithm to keep waters rigid rather than the slower SHAKE algorithm.

5.3.2 Harmonic restraint parameters

The following describes the parameters for the harmonic restraints feature of NAMD. For historical reasons the terminology of "harmonic constraints" has been carried over from X-PLOR. This feature allows a harmonic restraining force to be applied to any set of atoms in the simulation.

- **constraints** < are constraints active? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not harmonic constraints are active. If it is set to off, then no harmonic constraints are computed. If it is set to on, then harmonic constraints are calculated using the values specified by the parameters consref, conskfile, conskcol, and consexp.

- **consexp** < exponent for harmonic constraint energy function >
  Acceptable Values: positive, even integer
  Default Value: 2
  Description: Exponent to be use in the harmonic constraint energy function. This value must be a positive integer, and only even values really make sense. This parameter is used only if constraints is set to on.

- **consref** < PDB file containing constraint reference positions >
  Acceptable Values: UNIX file name
  Description: PDB file to use for reference positions for harmonic constraints. Each atom that has an active constraint will be constrained about the position specified in this file.

- **conskfile** < PDB file containing force constant values >
  Acceptable Values: UNIX filename
  Description: PDB file to use for force constants for harmonic constraints.

- **conskcol** < column of PDB file containing force constant >
  Acceptable Values: X, Y, Z, O, or B
  Description: Column of the PDB file to use for the harmonic constraint force constant. This parameter may specify any of the floating point fields of the PDB file, either X, Y, Z, occupancy, or beta-coupling (temperature-coupling). Regardless of which column is used, a value of 0 indicates that the atom should not be constrained. Otherwise, the value specified is used as the force constant for that atom's restraining potential.

- `constraintScaling` < scaling factor for harmonic constraint energy function >
  Acceptable Values: positive
  Default Value: 1.0
  Description: The harmonic constraint energy function is multiplied by this parameter, making it possible to gradually turn off constraints during equilibration. This parameter is used only if `constraints` is set to `on`.

- `selectConstraints` < Restrain only selected Cartesian components of the coordinates? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: This option is useful to restrain the positions of atoms to a plane or a line in space. If active, this option will ensure that only selected Cartesian components of the coordinates are restrained. E.g.: Restraining the positions of atoms to their current z values with no restraints in x and y will allow the atoms to move in the x-y plane while retaining their original z-coordinate. Restraining the x and y values will lead to free motion only along the z coordinate.

- `selectConstrX` < Restrain X components of coordinates >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Restrain the Cartesian x components of the positions.

- `selectConstrY` < Restrain Y components of coordinates >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Restrain the Cartesian y components of the positions.

- `selectConstrZ` < Restrain Z components of coordinates >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Restrain the Cartesian z components of the positions.

5.3.3 Fixed atoms parameters

Atoms may be held fixed during a simulation. NAMD avoids calculating most interactions in which all affected atoms are fixed unless `fixedAtomsForces` is specified.

- `fixedAtoms` < are there fixed atoms? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether or not fixed atoms are present.

- `fixedAtomsForces` < are forces between fixed atoms calculated? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether or not forces between fixed atoms are calculated. This option is required to turn fixed atoms off in the middle of a simulation. These forces will affect the pressure calculation, and you should leave this option off when using constant pressure if the coordinates of the fixed atoms have not been minimized. The use of constant pressure with significant numbers of fixed atoms is not recommended.

- `fixedAtomsFile` < PDB file containing fixed atom parameters >
  Acceptable Values: UNIX filename
  Default Value: `coordinates`
  Description: PDB file to use for the fixed atom flags for each atom. If this parameter is not specified, then the PDB file specified by `coordinates` is used.

- `fixedAtomsCol` < column of PDB containing fixed atom parameters >
  Acceptable Values: X, Y, Z, O, or B
  Default Value: O
  Description: Column of the PDB file to use for the containing fixed atom parameters for each atom. The coefficients can be read from any floating point column of the PDB file. A value of 0 indicates that the atom is not fixed.

6 Standard Minimization and Dynamics Parameters

6.1 Boundary Conditions

In addition to periodic boundary conditions, NAMD provides spherical and cylindrical boundary potentials to contain atoms in a given volume. To apply more general boundary potentials written in Tcl, use `tclBC` as described in Sec. 8.9.

6.1.1 Periodic boundary conditions

NAMD provides periodic boundary conditions in 1, 2 or 3 dimensions. The following parameters are used to define these boundary conditions.

- **cellBasisVector1** < basis vector for periodic boundaries (Å) >
  Acceptable Values: vector
  Default Value: 0 0 0
  Description: Specifies a basis vector for periodic boundary conditions.

- **cellBasisVector2** < basis vector for periodic boundaries (Å) >
  Acceptable Values: vector
  Default Value: 0 0 0
  Description: Specifies a basis vector for periodic boundary conditions.

- **cellBasisVector3** < basis vector for periodic boundaries (Å) >
  Acceptable Values: vector
  Default Value: 0 0 0
  Description: Specifies a basis vector for periodic boundary conditions.

- **cellOrigin** < center of periodic cell (Å) >
  Acceptable Values: position
  Default Value: 0 0 0
  Description: When position rescaling is used to control pressure, this location will remain constant. Also used as the center of the cell for wrapped output coordinates.

- **extendedSystem** < XSC file to read cell parameters from >
  Acceptable Values: file name
  Description: In addition to .coor and .vel output files, NAMD generates a .xsc (eXtended System Configuration) file which contains the periodic cell parameters and extended system variables, such as the strain rate in constant pressure simulations. Periodic cell parameters will be read from this file if this option is present, ignoring the above parameters.

- **XSTfile** < XST file to write cell trajectory to >
  Acceptable Values: file name
  Description: NAMD can also generate a .xst (eXtended System Trajectory) file which contains a record of the periodic cell parameters and extended system variables during the simulation. If `XSTfile` is defined, then `XSTfreq` must also be defined.

- **XSTfreq** < how often to append state to XST file >
  Acceptable Values: positive integer
  Description: Like the `DCDfreq` option, controls how often the extended system configuration will be appended to the XST file.

- **wrapWater** < wrap water coordinates around periodic boundaries? >
  Acceptable Values: on or off
  Default Value: off
  Description: Coordinates are normally output relative to the way they were read in. Hence, if part of a molecule crosses a periodic boundary it is not translated to the other side of the cell on output. This option alters this behavior for water molecules only.

- **wrapAll** < wrap all coordinates around periodic boundaries? >
  Acceptable Values: on or off
  Default Value: off
  Description: Coordinates are normally output relative to the way they were read in. Hence, if part of a molecule crosses a periodic boundary it is not translated to the other side of the cell on output. This option alters this behavior for all contiguous clusters of bonded atoms.

- **wrapNearest** < use nearest image to cell origin when wrapping coordinates? >
  Acceptable Values: on or off
  Default Value: off
  Description: Coordinates are normally wrapped to the diagonal unit cell centered on the origin. This option, combined with **wrapWater** or **wrapAll**, wraps coordinates to the nearest image to the origin, providing hexagonal or other cell shapes.

6.1.2   Spherical harmonic boundary conditions

NAMD provides spherical harmonic boundary conditions. These boundary conditions can consist of a single potential or a combination of two potentials. The following parameters are used to define these boundary conditions.

- **sphericalBC** < use spherical boundary conditions? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not spherical boundary conditions are to be applied to the system. If set to on, then **sphericalBCCenter**, **sphericalBCr1** and **sphericalBCk1** must be defined, and **sphericalBCexp1**, **sphericalBCr2**, **sphericalBCk2**, and **sphericalBCexp2** can optionally be defined.

- **sphericalBCCenter** < center of sphere (Å) >
  Acceptable Values: position
  Description: Location around which sphere is centered.

- **sphericalBCr1** < radius for first boundary condition (Å) >
  Acceptable Values: positive decimal
  Description: Distance at which the first potential of the boundary conditions takes effect. This distance is a radius from the center.

- **sphericalBCk1** < force constant for first potential >
  Acceptable Values: non-zero decimal
  Description: Force constant for the first harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- **sphericalBCexp1** < exponent for first potential >
  Acceptable Values: positive, even integer
  Default Value: 2
  Description: Exponent for first boundary potential. The only likely values to use are 2 and 4.

- **sphericalBCr2** < radius for second boundary condition (Å) >
  Acceptable Values: positive decimal
  Description: Distance at which the second potential of the boundary conditions takes effect. This distance is a radius from the center. If this parameter is defined, then **spericalBCk2** must also be defined.

- **sphericalBCk2** < force constant for second potential >
  Acceptable Values: non-zero decimal
  Description: Force constant for the second harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- **sphericalBCexp2** < exponent for second potential >
  Acceptable Values: positive, even integer
  Default Value: 2
  Description: Exponent for second boundary potential. The only likely values to use are 2 and 4.

6.1.3 Cylindrical harmonic boundary conditions

NAMD provides cylindrical harmonic boundary conditions. These boundary conditions can consist of a single potential or a combination of two potentials. The following parameters are used to define these boundary conditions.

- **cylindricalBC** < use cylindrical boundary conditions? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not cylindrical boundary conditions are to be applied to the system. If set to on, then **cylindricalBCCenter**, **cylindricalBCr1**, **cylindricalBCl1** and **cylindricalBCk1** must be defined, and **cylindricalBCAxis**, **cylindricalBCexp1**, **cylindricalBCr2**, **cylindricalBCl2**, **cylindricalBCk2**, and **cylindricalBCexp2** can optionally be defined.

- **cylindricalBCCenter** < center of cylinder (Å) >
  Acceptable Values: position
  Description: Location around which cylinder is centered.

- **cylindricalBCAxis** < axis of cylinder (Å) >
  Acceptable Values: x, y, or z
  Description: Axis along which cylinder is aligned.

- **cylindricalBCr1** < radius for first boundary condition (Å) >
  Acceptable Values: positive decimal
  Description: Distance at which the first potential of the boundary conditions takes effect along the non-axis plane of the cylinder.

- `cylindricalBCl1` < distance along cylinder axis for first boundary condition (Å) >
  Acceptable Values: positive decimal
  Description: Distance at which the first potential of the boundary conditions takes effect along the cylinder axis.

- `cylindricalBCk1` < force constant for first potential >
  Acceptable Values: non-zero decimal
  Description: Force constant for the first harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- `cylindricalBCexp1` < exponent for first potential >
  Acceptable Values: positive, even integer
  Default Value: 2
  Description: Exponent for first boundary potential. The only likely values to use are 2 and 4.

- `cylindricalBCr2` < radius for second boundary condition (Å) >
  Acceptable Values: positive decimal
  Description: Distance at which the second potential of the boundary conditions takes effect along the non-axis plane of the cylinder. If this parameter is defined, then `cylindricalBCl2` and `spericalBCk2` must also be defined.

- `cylindricalBCl2` < radius for second boundary condition (Å) >
  Acceptable Values: positive decimal
  Description: Distance at which the second potential of the boundary conditions takes effect along the cylinder axis. If this parameter is defined, then `cylindricalBCr2` and `spericalBCk2` must also be defined.

- `cylindricalBCk2` < force constant for second potential >
  Acceptable Values: non-zero decimal
  Description: Force constant for the second harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- `cylindricalBCexp2` < exponent for second potential >
  Acceptable Values: positive, even integer
  Default Value: 2
  Description: Exponent for second boundary potential. The only likely values to use are 2 and 4.

6.2 Energy Minimization

6.2.1 Conjugate gradient parameters

The default minimizer uses a sophisticated conjugate gradient and line search algorithm with much better performance than the older velocity quenching method. The method of conjugate gradients is used to select successive search directions (starting with the initial gradient) which eliminate repeated minimization along the same directions. Along each direction, a minimum is first bracketed (rigorously bounded) and then converged upon by either a golden section search, or, when possible, a quadratically convergent method using gradient information.

For most systems, it just works.

- **minimization** < Perform conjugate gradient energy minimization? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Turns efficient energy minimization `on` or `off`.

- **minTinyStep** < first initial step for line minimizer >
  Acceptable Values: positive decimal
  Default Value: 1.0e-6
  Description: If your minimization is immediately unstable, make this smaller.

- **minBabyStep** < max initial step for line minimizer >
  Acceptable Values: positive decimal
  Default Value: 1.0e-2
  Description: If your minimization becomes unstable later, make this smaller.

- **minLineGoal** < gradient reduction factor for line minimizer >
  Acceptable Values: positive decimal
  Default Value: 1.0e-4
  Description: Varying this might improve conjugate gradient performance.

6.2.2 Velocity quenching parameters

You can perform energy minimization using a simple quenching scheme. While this algorithm is not the most rapidly convergent, it is sufficient for most applications. There are only two parameters for minimization: one to activate minimization and another to specify the maximum movement of any atom.

- **velocityQuenching** < Perform old-style energy minimization? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Turns slow energy minimization `on` or `off`.

- **maximumMove** < maximum distance an atom can move during each step (Å) >
  Acceptable Values: positive decimal
  Default Value: $0.75 \times$ `cutoff`/`stepsPerCycle`
  Description: Maximum distance that an atom can move during any single timestep of minimization. This is to insure that atoms do not go flying off into space during the first few timesteps when the largest energy conflicts are resolved.

6.3 Dynamics

6.3.1 Timestep parameters

- **numsteps** < number of timesteps >
  Acceptable Values: positive integer
  Description: The number of simulation timesteps to be performed. An integer greater than 0 is acceptable. The total amount of simulation time is `numsteps` $\times$ `timestep`.

- **timestep** < timestep size (fs) >
  Acceptable Values: non-negative decimal

Default Value: 1.0

Description: The timestep size to use when integrating each step of the simulation. The value is specified in femtoseconds.

- `firsttimestep` < starting timestep value >
  Acceptable Values: non-negative integer
  Default Value: 0
  Description: The number of the first timestep. This value is typically used only when a simulation is a continuation of a previous simulation. In this case, rather than having the timestep restart at 0, a specific timestep number can be specified.

6.3.2 Initialization

- `temperature` < initial temperature (K) >
  Acceptable Values: positive decimal
  Description: Initial temperature value for the system. Using this option will generate a random velocity distribution for the initial velocities for all the atoms such that the system is at the desired temperature. Either the `temperature` or the `velocities/binvelocities` option must be defined to determine an initial set of velocities. Both options cannot be used together.

- `COMmotion` < allow initial center of mass motion? >
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: Specifies whether or not motion of the center of mass of the entire system is allowed. If this option is set to `no`, the initial velocities of the system will be adjusted to remove center of mass motion of the system. Note that this does not preclude later center-of-mass motion due to external forces such as random noise in Langevin dynamics, boundary potentials, and harmonic restraints.

- `seed` < random number seed >
  Acceptable Values: positive integer
  Default Value: pseudo-random value based on current UNIX clock time
  Description: Number used to seed the random number generator if `temperature` or `langevin` is selected. This can be used so that consecutive simulations produce the same results. If no value is specified, NAMD will choose a pseudo-random value based on the current UNIX clock time. The random number seed will be output during the simulation startup so that its value is known and can be reused for subsequent simulations. Note that if Langevin dynamics are used in a parallel simulation (i.e., a simulation using more than one processor) even using the same seed will *not* guarantee reproducible results.

6.3.3 Conserving momentum

- `zeroMomentum` < remove center of mass drift due to PME >
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: If enabled, the net momentum of the simulation and any resultant drift is removed before every full electrostatics step. This correction should conserve energy and

have minimal impact on parallel scaling. This feature should only be used for simulations that would conserve momentum except for the slight errors in PME. (Features such as fixed atoms, harmonic restraints, steering forces, and Langevin dynamics do not conserve momentum; use in combination with these features should be considered experimental.) Since the momentum correction is delayed, enabling outputMomenta will show a slight nonzero linear momentum but there should be no center of mass drift.

6.3.4 Multiple timestep parameters

To further reduce the cost of computing full electrostatics, NAMD uses a multiple timestepping integration scheme. In this scheme, the total force acting on each atom is broken into two pieces, a quickly varying local component and a slower long range component. The local force component is defined in terms of a *splitting function*. The local force component consists of all bonded and van der Waals interactions as well as that portion of electrostatic interactions for pairs that are separated by less than the local interaction distance determined by the splitting function. The long range component consists only of electrostatic interactions outside of the local interaction distance. Since the long range forces are slowly varying, they are not evaluated every timestep. Instead, they are evaluated every k timesteps, specified by the NAMD parameter `fullElectFrequency`. An impulse of k times the long range force is applied to the system every k timesteps (i.e., the r-RESPA integrator is used). For appropriate values of k, it is believed that the error introduced by this infrequent evaluation is modest compared to the error already incurred by the use of the numerical (Verlet) integrator. Improved methods for incorporating these long range forces are currently being investigated, with the intention of improving accuracy as well as reducing the frequency of long range force evaluations.

In the scheme described above, the van der Waals forces are still truncated at the local interaction distance. Thus, the van der Waals cutoff distance forms a lower limit to the local interaction distance. While this is believed to be sufficient, there are investigations underway to remove this limitation and provide full van der Waals calculations in $\mathcal{O}(N)$ time as well.

One of the areas of current research being studied using NAMD is the exploration of better methods for performing multiple timestep integration. Currently the only available method is the impulse-based Verlet-I or r-RESPA method which is stable for timesteps up to 4 fs for long-range electrostatic forces, 2 fs for short-range nonbonded forces, and 1 fs for bonded forces Setting `rigid all` (i.e., using SHAKE) increases these timesteps to 6 fs, 2 fs, and 2 fs respectively but eliminates bond motion for hydrogen. The mollified impulse method (MOLLY) reduces the resonance which limits the timesteps and thus increases these timesteps to 6 fs, 2 fs, and 1 fs while retaining all bond motion.

- `fullElectFrequency` < number of timesteps between full electrostatic evaluations >
  Acceptable Values: positive integer factor of `stepspercycle`
  Default Value: `nonbondedFreq`
  Description: This parameter specifies the number of timesteps between each full electrostatics evaluation. It is recommended that `fullElectFrequency` be chosen so that the product of `fullElectFrequency` and `timestep` does not exceed 4.0 unless `rigidBonds all` or `molly on` is specified, in which case the upper limit is perhaps doubled.

- `nonbondedFreq` < timesteps between nonbonded evaluation >
  Acceptable Values: positive integer factor of `fullElectFrequency`

Default Value: 1

Description: This parameter specifies how often short-range nonbonded interactions should be calculated. Setting `nonbondedFreq` between 1 and `fullElectFrequency` allows triple timestepping where, for example, one could evaluate bonded forces every 1 fs, short-range nonbonded forces every 2 fs, and long-range electrostatics every 4 fs.

- `MTSAlgorithm` < MTS algorithm to be used >
  Acceptable Values: `impulse/verletI` or `constant/naive`
  Default Value: `impulse`
  Description: Specifies the multiple timestep algorithm used to integrate the long and short range forces. `impulse/verletI` is the same as r-RESPA. `constant/naive` is the stale force extrapolation method.

- `longSplitting` < how should long and short range forces be split? >
  Acceptable Values: `xplor`, `c1`
  Default Value: `c1`
  Description: Specifies the method used to split electrostatic forces between long and short range potentials. The `xplor` option uses the X-PLOR shifting function, and the `c1` splitting uses the following C^1 continuous shifting function [26]:

  $$SW(r_{ij}) = 0 \text{ if } |\vec{r}_{ij}| > R_{off}$$
  $$SW(r_{ij}) = 1 \text{ if } |\vec{r}_{ij}| \le R_{on}$$
  $$\text{if } R_{off} > |\vec{r}_{ij}| \ge R_{on}$$

  where

  R_{on} is a constant defined using the configuration value `switchdist`

  R_{off} is specified using the configuration value `cutoff`

- `molly` < use mollified impulse method (MOLLY)? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: This method eliminates the components of the long range electrostatic forces which contribute to resonance along bonds to hydrogen atoms, allowing a fullElectFrequency of 6 (vs. 4) with a 1 fs timestep without using `rigidBonds all`. You may use `rigidBonds water` but using `rigidBonds all` with MOLLY makes no sense since the degrees of freedom which MOLLY protects from resonance are already frozen.

- `mollyTolerance` < allowable error for MOLLY >
  Acceptable Values: positive decimal
  Default Value: 0.00001
  Description: Convergence criterion for MOLLY algorithm.

- `mollyIterations` < maximum MOLLY iterations >
  Acceptable Values: positive integer
  Default Value: 100
  Description: Maximum number of iterations for MOLLY algorithm.

6.4 Temperature Control and Equilibration

6.4.1 Langevin dynamics parameters

NAMD is capable of performing Langevin dynamics, where additional damping and random forces are introduced to the system. This capability is based on that implemented in X-PLOR which is detailed in the X-PLOR *User's Manual* [9], although a different integrator is used.

- **langevin** < use Langevin dynamics? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not Langevin dynamics active. If set to on, then the parameter langevinTemp must be set and the parameters langevinFile and langevinCol can optionally be set to control the behavior of this feature.

- **langevinTemp** < temperature for Langevin calculations (K) >
  Acceptable Values: positive decimal
  Description: Temperature to which atoms affected by Langevin dynamics will be adjusted. This temperature will be roughly maintained across the affected atoms through the addition of friction and random forces.

- **langevinDamping** < damping coefficient for Langevin dynamics (1/ps) >
  Acceptable Values: positive decimal
  Default Value: per-atom values from PDB file
  Description: Langevin coupling coefficient to be applied to all atoms (unless langevinHydrogen is off, in which case only non-hydrogen atoms are affected). If not given, a PDB file is used to obtain coefficients for each atom (see langevinFile and langevinCol below).

- **langevinHydrogen** < Apply Langevin dynamics to hydrogen atoms? >
  Acceptable Values: on or off
  Default Value: on
  Description: If langevinDamping is set then setting langevinHydrogen to off will turn off Langevin dynamics for hydrogen atoms. This parameter has no effect if Langevin coupling coefficients are read from a PDB file.

- **langevinFile** < PDB file containing Langevin parameters >
  Acceptable Values: UNIX filename
  Default Value: coordinates
  Description: PDB file to use for the Langevin coupling coefficients for each atom. If this parameter is not specified, then the PDB file specified by coordinates is used.

- **langevinCol** < column of PDB from which to read coefficients >
  Acceptable Values: X, Y, Z, O, or B
  Default Value: O
  Description: Column of the PDB file to use for the Langevin coupling coefficients for each atom. The coefficients can be read from any floating point column of the PDB file. A value of 0 indicates that the atom will remain unaffected.

6.4.2 Temperature coupling parameters

NAMD is capable of performing temperature coupling, in which forces are added or reduced to simulate the coupling of the system to a heat bath of a specified temperature. This capability is based on that implemented in X-PLOR which is detailed in the X-PLOR *User's Manual* [9].

- **tCouple** < perform temperature coupling? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not temperature coupling is active. If set to on, then the parameter tCoupleTemp must be set and the parameters tCoupleFile and tCoupleCol can optionally be set to control the behavior of this feature.

- **tCoupleTemp** < temperature for heat bath (K) >
  Acceptable Values: positive decimal
  Description: Temperature to which atoms affected by temperature coupling will be adjusted. This temperature will be roughly maintained across the affected atoms through the addition of forces.

- **tCoupleFile** < PDB file with tCouple parameters >
  Acceptable Values: UNIX filename
  Default Value: coordinates
  Description: PDB file to use for the temperature coupling coefficient for each atom. If this parameter is not specified, then the PDB file specified by coordinates is used.

- **tCoupleCol** < column of PDB from which to read coefficients >
  Acceptable Values: X, Y, Z, O, or B
  Default Value: O
  Description: Column of the PDB file to use for the temperature coupling coefficient for each atom. This value can be read from any floating point column of the PDB file. A value of 0 indicates that the atom will remain unaffected.

6.4.3 Temperature rescaling parameters

NAMD allows equilibration of a system by means of temperature rescaling. Using this method, all of the velocities in the system are periodically rescaled so that the entire system is set to the desired temperature. The following parameters specify how often and to what temperature this rescaling is performed.

- **rescaleFreq** < number of timesteps between temperature rescaling >
  Acceptable Values: positive integer
  Description: The equilibration feature of NAMD is activated by specifying the number of timesteps between each temperature rescaling. If this value is given, then the rescaleTemp parameter must also be given to specify the target temperature.

- **rescaleTemp** < temperature for equilibration (K) >
  Acceptable Values: positive decimal
  Description: The temperature to which all velocities will be rescaled every rescaleFreq timesteps. This parameter is valid only if rescaleFreq has been set.

6.4.4 Temperature reassignment parameters

NAMD allows equilibration of a system by means of temperature reassignment. Using this method, all of the velocities in the system are periodically reassigned so that the entire system is set to the desired temperature. The following parameters specify how often and to what temperature this reassignment is performed.

- **reassignFreq** < number of timesteps between temperature reassignment >
  Acceptable Values: positive integer
  Description: The equilibration feature of NAMD is activated by specifying the number of timesteps between each temperature reassignment. If this value is given, then the **reassignTemp** parameter must also be given to specify the target temperature.

- **reassignTemp** < temperature for equilibration (K) >
  Acceptable Values: positive decimal
  Default Value: **temperature** if set, otherwise none
  Description: The temperature to which all velocities will be reassigned every **reassignFreq** timesteps. This parameter is valid only if **reassignFreq** has been set.

- **reassignIncr** < temperature increment for equilibration (K) >
  Acceptable Values: decimal
  Default Value: 0
  Description: In order to allow simulated annealing or other slow heating/cooling protocols, **reassignIncr** will be added to **reassignTemp** after each reassignment. (Reassignment is carried out at the first timestep.) The **reassignHold** parameter may be set to limit the final temperature. This parameter is valid only if **reassignFreq** has been set.

- **reassignHold** < holding temperature for equilibration (K) >
  Acceptable Values: positive decimal
  Description: The final temperature for reassignment when **reassignIncr** is set; **reassignTemp** will be held at this value once it has been reached. This parameter is valid only if **reassignIncr** has been set.

6.5 Pressure Control

Constant pressure simulation (and pressure calculation) require periodic boundary conditions. Pressure is controlled by dynamically adjusting the size of the unit cell and rescaling all atomic coordinates (other than those of fixed atoms) during the simulation.

Pressure values in NAMD output are in bar. PRESSURE is the pressure calculated based on individual atoms, while GPRESSURE incorporates hydrogen atoms into the heavier atoms to which they are bonded, producing smaller fluctuations. The TEMPAVG, PRESSAVG, and GPRESSAVG are the average of temperature and pressure values since the previous ENERGY output; for the first step in the simulation they will be identical to TEMP, PRESSURE, and GPRESSURE.

The phenomenological pressure of bulk matter reflects averaging in both space and time of the sum of a large positive term (the kinetic pressure, nRT/V), and a large cancelling negative term (the static pressure). The instantaneous pressure of a simulation cell as simulated by NAMD will have mean square fluctuations (according to David Case quoting Section 114 of *Statistical Physics* by Landau and Lifshitz) of $kT/(V\beta)$, where β is the compressibility, which is RMS of roughly 100

bar for a 10,000 atom biomolecular system. Much larger fluctuations are regularly observed in practice.

The instantaneous pressure for a biomolecular system is well defined for "internal" forces that are based on particular periodic images of the interacting atoms, conserve momentum, and are translationally invariant. When dealing with externally applied forces such as harmonic constraints, fixed atoms, and various steering forces, NAMD bases its pressure calculation on the relative positions of the affected atoms in the input coordinates and assumes that the net force will average to zero over time. For time periods during with the net force is non-zero, the calculated pressure fluctuations will include a term proportional to the distance to the affected from the user-defined cell origin. A good way to observe these effects and to confirm that pressure for external forces is handled reasonably is to run a constant volume cutoff simulation in a cell that is larger than the molecular system by at least the cutoff distance; the pressure for this isolated system should average to zero over time.

Because NAMD's impluse-basd multiple timestepping system alters the balance between bonded and non-bonded forces from every timestep to an average balance over two steps, the calculated pressure on even and odd steps will be different. The PRESSAVG and GPRESSAVG fields provide the average over the non-printed intermediate steps. If you print energies on every timestep you will see the effect clearly in the PRESSURE field.

The following options affect all pressure control methods.

- **useGroupPressure** < group or atomic quantities >
  Acceptable Values: yes or no
  Default Value: no
  Description: Pressure can be calculated using either the atomic virial and kinetic energy (the default) or a hydrogen-group based pseudo-molecular virial and kinetic energy. The latter fluctuates less and is required in conjunction with rigidBonds (SHAKE).

- **useFlexibleCell** < anisotropic cell fluctuations >
  Acceptable Values: yes or no
  Default Value: no
  Description: NAMD allows the three orthogonal dimensions of the periodic cell to fluctuate independently when this option is enabled.

- **useConstantRatio** < constant shape in first two cell dimensions >
  Acceptable Values: yes or no
  Default Value: no
  Description: When enabled, NAMD keeps the ratio of the unit cell in the x-y plane constant while allowing fluctuations along all axes. The **useFlexibleCell** option is required for this option.

- **useConstantArea** < constant area and normal pressure conditions >
  Acceptable Values: yes or no
  Default Value: no
  Description: When enabled, NAMD keeps the dimension of the unit cell in the x-y plane constant while allowing fluctuations along the z axis. This is not currently implemented in Berendsen's method.

6.5.1 Berendsen pressure bath coupling

NAMD provides constant pressure simulation using Berendsen's method. The following parameters are used to define the algorithm.

- `BerendsenPressure` < use Berendsen pressure bath coupling? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether or not Berendsen pressure bath coupling is active. If set to **on**, then the parameters `BerendsenPressureTarget`, `BerendsenPressureCompressibility` and `BerendsenPressureRelaxationTime` must be set and the parameter `BerendsenPressureFreq` can optionally be set to control the behavior of this feature.

- `BerendsenPressureTarget` < target pressure (bar) >
  Acceptable Values: positive decimal
  Description: Specifies target pressure for Berendsen's method. A typical value would be 1.01325 bar, atmospheric pressure at sea level.

- `BerendsenPressureCompressibility` < compressibility (bar^{-1}) >
  Acceptable Values: positive decimal
  Description: Specifies compressibility for Berendsen's method. A typical value would be 4.57E-5 bar^{-1}, corresponding to liquid water. The higher the compressibility, the more volume will be adjusted for a given pressure difference. The compressibility and the relaxation time appear only as a ratio in the dynamics, so a larger compressibility is equivalent to a smaller relaxation time.

- `BerendsenPressureRelaxationTime` < relaxation time (fs) >
  Acceptable Values: positive decimal
  Description: Specifies relaxation time for Berendsen's method. If the instantaneous pressure did not fluctuate randomly during a simulation and the compressibility estimate was exact then the inital pressure would decay exponentially to the target pressure with this time constant. Having a longer relaxation time results in more averaging over pressure measurements and hence smaller fluctuations in the cell volume. A reasonable choice for relaxation time would be 100 fs. The compressibility and the relaxation time appear only as a ratio in the dynamics, so a larger compressibility is equivalent to a smaller relaxation time.

- `BerendsenPressureFreq` < how often to rescale positions >
  Acceptable Values: positive multiple of `nonbondedFrequency` and `fullElectFrequency`
  Default Value: `nonbondedFrequency` or `fullElectFrequency` if used
  Description: Specifies number of timesteps between position rescalings for Berendsen's method. Primarily to deal with multiple timestepping integrators, but also to reduce cell volume fluctuations, cell rescalings can occur on a longer interval. This could reasonably be between 1 and 20 timesteps, but the relaxation time should be at least ten times larger.

6.5.2 Nosé-Hoover Langevin piston pressure control

NAMD provides constant pressure simulation using a modified Nosé-Hoover method in which Langevin dynamics is used to control fluctuations in the barostat. This method should be combined

with a method of temperature control, such as Langevin dynamics, in order to simulate the NPT ensemble.

The Langevin piston Nose-Hoover method in NAMD is a combination of the Nose-Hoover constant pressure method as described in GJ Martyna, DJ Tobias and ML Klein, "Constant pressure molecular dynamics algorithms", J. Chem. Phys 101(5), 1994, with piston fluctuation control implemented using Langevin dynamics as in SE Feller, Y Zhang, RW Pastor and BR Brooks, "Constant pressure molecular dynamics simulation: The Langevin piston method", J. Chem. Phys. 103(11), 1995.

The equations of motion are:

$$
\begin{aligned}
r' &= p/m + e'r \\
p' &= F - e'p - gp + R \\
V' &= 3Ve' \\
e'' &= 3V/W(P - P_0) - g_e e' + R_e/W \\
W &= 3N\tau^2 kT \\
<R^2> &= 2mgkT/h \\
\tau &= \text{oscillationperiod} \\
<R_e^2> &= 2Wg_e kT/h
\end{aligned}
$$

Here, W is the mass of piston, R is noise on atoms, and R_e is the noise on the piston.

The user specifies the desired pressure, oscillation and decay times of the piston, and temperature of the piston. The compressibility of the system is not required. In addition, the user specifies the damping coefficients and temperature of the atoms for Langevin dynamics.

The following parameters are used to define the algorithm.

- LangevinPiston < use Langevin piston pressure control? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not Langevin piston pressure control is active. If set to on, then the parameters LangevinPistonTarget, LangevinPistonPeriod, LangevinPistonDecay and LangevinPistonTemp must be set.

- LangevinPistonTarget < target pressure (bar) >
  Acceptable Values: positive decimal
  Description: Specifies target pressure for Langevin piston method. A typical value would be 1.01325 bar, atmospheric pressure at sea level.

- LangevinPistonPeriod < oscillation period (fs) >
  Acceptable Values: positive decimal
  Description: Specifies barostat oscillation time scale for Langevin piston method. If the instantaneous pressure did not fluctuate randomly during a simulation and the decay time was infinite (no friction) then the cell volume would oscillate with this angular period. Having a longer period results in more averaging over pressure measurements and hence slower fluctuations in the cell volume. A reasonable choice for the piston period would be 200 fs.

- LangevinPistonDecay < damping time scale (fs) >
  Acceptable Values: positive decimal

Description: Specifies barostat damping time scale for Langevin piston method. A value larger than the piston period would result in underdamped dynamics (decaying ringing in the cell volume) while a smaller value approaches exponential decay as in Berendsen's method above. A smaller value also corresponds to larger random forces with increased coupling to the Langevin temperature bath. Typically this would be chosen equal to or smaller than the piston period, such as 100 fs.

- `LangevinPistonTemp` < noise temperature (K) >
  Acceptable Values: positive decimal
  Description: Specifies barostat noise temperature for Langevin piston method. This should be set equal to the target temperature for the chosen method of temperature control.

- `SurfaceTensionTarget` < Surface tension target (dyn/cm) >
  Acceptable Values: decimal
  Default Value: 0.0
  Description: Specifies surface tension target. Must be used with `useFlexibleCell` and periodic boundary conditions. The pressure specified in `LangevinPistonTarget` becomes the pressure along the z axis, and surface tension is applied in the x-y plane.

- `StrainRate` < initial strain rate >
  Acceptable Values: decimal triple (x y z)
  Default Value: 0. 0. 0.
  Description: Optionally specifies the initial strain rate for pressure control. Is overridden by value read from file specified with `extendedSystem`. There is typically no reason to set this parameter.

- `ExcludeFromPressure` < Should some atoms be excluded from pressure rescaling? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether or not to exclude some atoms from pressure rescaling. The coordinates and velocites of such atoms are not rescaled during constant pressure simulations, though they do contribute to the virial calculation. May be useful for membrane protein simulation. EXPERIMENTAL.

- `ExcludeFromPressureFile` < File specifying excluded atoms >
  Acceptable Values: PDB file
  Default Value: coordinates file
  Description: PDB file with one column specifying which atoms to exclude from pressure rescaling. Specify 1 for excluded and 0 for not excluded.

- `ExcludeFromPressureCol` < Column in PDB file for specifying excluded atoms >
  Acceptable Values: O, B, X, Y, or Z
  Default Value: O
  Description: Specifies which column of the pdb file to check for excluded atoms.

7 Performance Tuning

7.1 Non-bonded interaction distance-testing

The last critical parameter for non-bonded interaction calculations is the parameter `pairlistdist`. To reduce the cost of performing the non-bonded interactions, NAMD uses a *non-bonded pair list* which contained all pairs of atoms for which non-bonded interactions should be calculated. Performing the search for pairs of atoms that should have their interactions calculated is an expensive operation. Thus, the pair list is only calculated periodically, at least once per cycle. Unfortunately, pairs of atoms move relative to each other during the steps between preparation of the pair list. Because of this, if the pair list were built to include only those pairs of atoms that are within the cutoff distance when the list is generated, it would be possible for atoms to drift closer together than the cutoff distance during subsequent timesteps and yet not have their non-bonded interactions calculated.

Let us consider a concrete example to better understand this. Assume that the pairlist is built once every ten timesteps and that the cutoff distance is 8.0 Å. Consider a pair of atoms A and B that are 8.1 Å apart when the pairlist is built. If the pair list includes only those atoms within the cutoff distance, this pair would not be included in the list. Now assume that after five timesteps, atoms A and B have moved to only 7.9 Å apart. A and B are now within the cutoff distance of each other, and should have their non-bonded interactions calculated. However, because the non-bonded interactions are based solely on the pair list and the pair list will not be rebuilt for another five timesteps, this pair will be ignored for five timesteps causing energy not to be conserved within the system.

To avoid this problem, the parameter `pairlistdist` allows the user to specify a distance greater than the `cutoff` distance for pairs to be included in the pair list, as shown in Figure 4. Pairs that are included in the pair list but are outside the cutoff distance are simply ignored. So in the above example, if the `pairlistdist` were set to 10.0 Å, then the atom pair A and B would be included in the pair list, even though the pair would initially be ignored because they are further apart than the cutoff distance. As the pair moved closer and entered the cutoff distance, because the pair was already in the pair list, the non-bonded interactions would immediately be calculated and energy conservation would be preserved. The value of `pairlistdist` should be chosen such that no atom pair moves more than `pairlistdist − cutoff` in one cycle. This will insure energy conservation and efficiency.

The `pairlistdist` parameter is also used to determine the minimum patch size. Unless the `splitPatch` parameter is explicitly set to `position`, hydrogen atoms will be placed on the same patch as the "mother atom" to which they are bonded. These *hydrogen groups* are then distance tested against each other using only a cutoff increased by the the value of the `hgroupCutoff` parameter. The size of the patches is also increased by this amount. NAMD functions correctly even if a hydrogen atom and its mother atom are separated by more than half of `hgroupCutoff` by breaking that group into its individual atoms for distance testing. Margin violation warning messages are printed if an atom moves outside of a safe zone surrounding the patch to which it is assigned, indicating that `pairlistdist` should be increased in order for forces to be calculated correctly and energy to be conserved.

Margin violations mean that atoms that are in non-neighboring patches may be closer than the cutoff distance apart. This may sometimes happen in constant pressure simulations when the cell shrinks (since the patch grid remains the same size). The workaround is to increase the margin parameter so that the simulation starts with fewer, larger patches. Restarting the simulation will
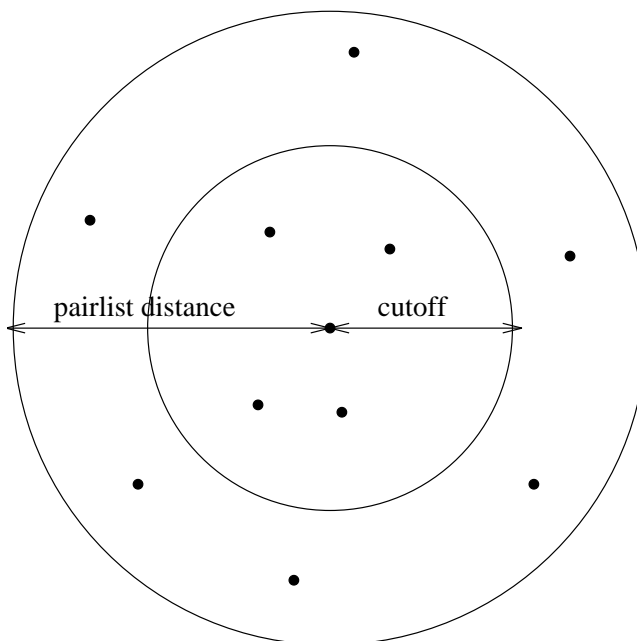
Figure 4: Depiction of the difference between the cutoff distance and the pair list distance. The pair list distance specifies a sphere that is slightly larger than that of the cutoff so that pairs are allowed to move in and out of the cutoff distance without causing energy conservation to be disturbed.

also regenerate the patch grid.

In rare special circumstances atoms that are involved in bonded terms (bonds, angles, dihedrals, or impropers) or nonbonded exclusions (especially implicit exclusions due to bonds) will be placed on non-neighboring patches because they are more than the cutoff distance apart. This can result in the simulation dying with a message of "bad global exclusion count". If an "atoms moving too fast; simulation has become unstable", "bad global exclusion count", or similar error happens on the first timestep then there is likely something very wrong with the input coordinates, such as the atoms with uninitialized coordinates or different atom orders in the PSF and PDB file. Looking at the system in VMD will often reveal an abnormal structure. Be aware that the atom IDs in the "Atoms moving too fast" error message are 1-based, while VMD's atom indices are 0-based. If an "atoms moving too fast; simulation has become unstable", "bad global exclusion count", or similar error happens later in the simulation then the dynamics have probably become unstable, resulting in the system "exploding" apart. Energies printed at every timestep should show an exponential increase. This may be due to a timestep that is too long, or some other strange feature. Saving a trajectory of every step and working backwards in can also sometimes reveal the origin of the instability.

- **pairlistdist** < distance between pairs for inclusion in pair lists (Å) >
  Acceptable Values: positive decimal ≥ `cutoff`
  Default Value: `cutoff`
  Description: A pair list is generated `pairlistsPerCycle` times each cycle, containing pairs of atoms for which electrostatics and van der Waals interactions will be calculated. This parameter is used when `switching` is set to `on` to specify the allowable distance between atoms for inclusion in the pair list. This parameter is equivalent to the X-PLOR parameter

69

CUTNb. If no atom moves more than `pairlistdist−cutoff` during one cycle, then there will be no jump in electrostatic or van der Waals energies when the next pair list is built. Since such a jump is unavoidable when truncation is used, this parameter may only be specified when `switching` is set to `on`. If this parameter is not specified and `switching` is set to `on`, the value of `cutoff` is used. A value of at least one greater than `cutoff` is recommended.

- `stepspercycle` < timesteps per cycle >
  Acceptable Values: positive integer
  Default Value: 20
  Description: Number of timesteps in each cycle. Each cycle represents the number of timesteps between atom reassignments. For more details on non-bonded force evaluation, see Section 5.2.

- `splitPatch` < how to assign atoms to patches >
  Acceptable Values: `position` or `hydrogen`
  Default Value: `hydrogen`
  Description: When set to `hydrogen`, hydrogen atoms are kept on the same patch as their parents, allowing faster distance checking and rigid bonds.

- `hgroupCutoff` (Å) < used for group-based distance testing >
  Acceptable Values: positive decimal
  Default Value: 2.5
  Description: This should be set to twice the largest distance which will ever occur between a hydrogen atom and its mother. Warnings will be printed if this is not the case. This value is also added to the margin.

- `margin` < extra length in patch dimension (Å) >
  Acceptable Values: positive decimal
  Default Value: 0.0
  Description: An internal tuning parameter used in determining the size of the cubes of space with which NAMD uses to partition the system. The value of this parameter will not change the physical results of the simulation. Unless you are very motivated to get the *very* best possible performance, just leave this value at the default.

- `pairlistMinProcs` < min procs for pairlists >
  Acceptable Values: positive integer
  Default Value: 1
  Description: Pairlists may consume a large amount of memory as atom counts, densities, and cutoff distances increase. Since this data is distributed across processors it is normally only problematic for small processor counts. Set pairlistMinProcs to the smallest number of processors on which the simulation can fit into memory when pairlists are used.

- `pairlistsPerCycle` < regenerate x times per cycle >
  Acceptable Values: positive integer
  Default Value: 2
  Description: Rather than only regenerating the pairlist at the beginning of a cycle, regenerate multiple times in order to better balance the costs of atom migration, pairlist generation, and larger pairlists.

- `outputPairlists` < how often to print warnings >
  Acceptable Values: non-negative integer
  Default Value: 0
  Description: If an atom moves further than the pairlist tolerance during a simulation (initially (pairlistdist - cutoff)/2 but refined during the run) any pairlists covering that atom are invalidated and temporary pairlists are used until the next full pairlist regeneration. All interactions are calculated correctly, but efficiency may be degraded. Enabling outputPairlists will summarize these pairlist violation warnings periodically during the run.

- `pairlistShrink` < tol *= (1 - x) on regeneration >
  Acceptable Values: non-negative decimal
  Default Value: 0.01
  Description: In order to maintain validity for the pairlist for an entire cycle, the pairlist tolerance (the distance an atom can move without causing the pairlist to be invalidated) is adjusted during the simulation. Every time pairlists are regenerated the tolerance is reduced by this fraction.

- `pairlistGrow` < tol *= (1 + x) on trigger >
  Acceptable Values: non-negative decimal
  Default Value: 0.01
  Description: In order to maintain validity for the pairlist for an entire cycle, the pairlist tolerance (the distance an atom can move without causing the pairlist to be invalidated) is adjusted during the simulation. Every time an atom exceeds a trigger criterion that is some fraction of the tolerance distance, the tolerance is increased by this fraction.

- `pairlistTrigger` < trigger is atom beyond (1 - x) * tol >
  Acceptable Values: non-negative decimal
  Default Value: 0.3
  Description: The goal of pairlist tolerance adjustment is to make pairlist invalidations rare while keeping the tolerance as small as possible for best performance. Rather than monitoring the (very rare) case where atoms actually move more than the tolerance distance, we reduce the trigger tolerance by this fraction. The tolerance is increased whenever the trigger tolerance is exceeded, as specified by pairlistGrow.

8   User Defined Forces

There are several ways to apply external forces to simulations with NAMD. These are described below.

8.1   Constant Forces

NAMD provides the ability to apply constant forces to some atoms. There are two parameters that control this feature.

- **constantForce** < Apply constant forces? >
  Acceptable Values: yes or no
  Default Value: no
  Description: Specifies whether or not constant forces are applied.

- **consForceFile** < PDB file containing forces to be applied >
  Acceptable Values: UNIX filename
  Description: The X, Y, Z and occupancy (O) fields of this file are read to determine the constant force vector of each atom, which is (X,Y,Z)*O, in unit of kcal/(mol*Å). The occupancy (O) serves as a scaling factor, which could expand the range of the force applied. (One may be unable to record very large or very small numbers in the data fields of a PDB file due to limited space). Zero forces are ignored.

  Specifying `consforcefile` is optional; constant forces may be specifed or updated between runs by using the consForceConfig command.

- **consForceScaling** < Scaling factor for constant forces >
  Acceptable Values: decimal
  Default Value: 1.0
  Description: Scaling factor by which constant forces are multiplied. May be changed between run commands.

8.2   External Electric Field

NAMD provides the ability to apply a constant electric field to the molecular system being simulated. Energy due to the external field will be reported in the MISC column and may be discontinuous in simulations using periodic boundary conditions if, for example, a charged hydrogen group moves outside of the central cell. There are two parameters that control this feature.

- **eFieldOn** < apply electric field? >
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: Specifies whether or not an electric field is applied.

- **eField** < electric field vector >
  Acceptable Values: vector of decimals (x y z)
  Description: Vector which describes the electric field to be applied.   Units are kcal/(mol Å e), which is natural for simulations. This parameter may be changed between `run` commands, allowing a square wave or other approximate wave form to be applied.

8.3 Moving Constraints

Moving constraints feature works in conjunction with the Harmonic Constraints (see an appropriate section of the User's guide). The reference positions of all constraints will move according to

$$\vec{r}(t) = \vec{r}_0 + \vec{v}t. \tag{8}$$

A velocity vector $\vec{v}$ (`movingConsVel`) needs to be specified.

The way the moving constraints work is that the moving reference position is calculated every integration time step using Eq. 8, where $\vec{v}$ is in Å/timestep, and t is the current timestep (i.e., `firstTimestep` plus however many timesteps have passed since the beginning of NAMD run). Therefore, one should be careful when restarting simulations to appropriately update the `firstTimestep` parameter in the NAMD configuration file or the reference position specified in the reference PDB file.

NOTE: NAMD actually calculates the constraints potential with $U = k(x - x_0)^d$ and the force with $F = dk(x - x_0)$, where d is the exponent `consexp`. The result is that if one specifies some value for the force constant k in the PDB file, effectively, the force constant is $2k$ in calculations. This caveat was removed in SMD feature.

The following parameters describe the parameters for the moving harmonic constraint feature of NAMD.

- `movingConstraints` < Are moving constraints active >
  Acceptable Values: on or off
  Default Value: off
  Description: Should moving restraints be applied to the system. If set to **on**, then `movingConsVel` must be defined. May not be used with `rotConstraints`.

- `movingConsVel` < Velocity of the reference position movement >
  Acceptable Values: vector in Å/timestep
  Description: The velocity of the reference position movement. Gives both absolute value and direction

8.4 Rotating Constraints

The constraints parameters are specified in the same manner as for usual (static) harmonic constraints. The reference positions of all constrained atoms are then rotated with a given angular velocity about a given axis. If the force constant of the constraints is sufficiently large, the constrained atoms will follow their reference positions.

A rotation matrix M about the axis unit vector v is calculated every timestep for the angle of rotation corresponding to the current timestep. angle $= \Omega t$, where Ω is the angular velocity of rotation.

From now on, all quantities are 3D vectors, except the matrix M and the force constant K.

The current reference position R is calculated from the initial reference position R_0 (at $t = 0$), $R = M(R_0 - P) + P$, where P is the pivot point.

Coordinates of point N can be found as $N = P + ((R - P) \cdot v)v$. Normal from the atom pos to the axis is, similarly, normal $= (P + ((X - P) \cdot v)v) - X$ The force is, as usual, $F = K(R - X)$; This is the force applied to the atom in NAMD (see below). NAMD does not know anything about the torque applied. However, the torque applied to the atom can be calculated as a vector

product torque $= F \times normal$ Finally, the torque applied to the atom with respect to the axis is the projection of the torque on the axis, i.e., $torque_{proj} = torque \cdot v$

If there are atoms that have to be constrained, but not moved, this implementation is not suitable, because it will move *all* reference positions.

Only one of the moving and rotating constraints can be used at a time.

Using very soft springs for rotating constraints leads to the system lagging behind the reference positions, and then the force is applied along a direction different from the "ideal" direction along the circular path.

Pulling on N atoms at the same time with a spring of stiffness K amounts to pulling on the whole system by a spring of stiffness NK, so the overall behavior of the system is as if you are pulling with a very stiff spring if N is large.

In both moving and rotating constraints the force constant that you specify in the constraints pdb file is multiplied by 2 for the force calculation, i.e., if you specified $K = 0.5$ kcal/mol/Å^2 in the pdb file, the force actually calculated is $F = 2K(R - X) = 1$ kcal/mol/Å^2 $(R - X)$. SMD feature of namd2 does the calculation without multiplication of the force constant specified in the config file by 2.

- `rotConstraints` $<$ Are rotating constraints active $>$
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Should rotating restraints be applied to the system. If set to `on`, then `rotConsAxis`, `rotConsPivot` and `rotConsVel` must be defined. May not be used with `movingConstraints`.

- `rotConsAxis` $<$ Axis of rotation $>$
  Acceptable Values: vector (may be unnormalized)
  Description: Axis of rotation. Can be any vector. It gets normalized before use. If the vector is 0, no rotation will be performed, but the calculations will still be done.

- `rotConsPivot` $<$ Pivot point of rotation $>$
  Acceptable Values: position in Å
  Description: Pivot point of rotation. The rotation axis vector only gives the direction of the axis. Pivot point places the axis in space, so that the axis goes through the pivot point.

- `rotConsVel` $<$ Angular velocity of rotation $>$
  Acceptable Values: rate in degrees per timestep
  Description: Angular velocity of rotation, degrees/timestep.

8.5 Targeted Molecular Dynamics (TMD)

In TMD, subset of atoms in the simulation is guided towards a final 'target' structure by means of steering forces. At each timestep, the RMS distance between the current coordinates and the target structure is computed (after first aligning the target structure to the current coordinates). The force on each atom is given by the gradient of the potential

$$U_{TMD} = \frac{1}{2} \frac{k}{N} \left[RMS(t) - RMS^*(t) \right]^2 \qquad (9)$$

where $RMS(t)$ is the instantaneous best-fit RMS distance of the current coordinates from the target coordinates, and $RMS^*(t)$ evolves linearly from the initial RMSD at the first TMD step to

the final RMSD at the last TMD step. The spring constant k is scaled down by the number N of targeted atoms.

- **TMD** $<$ Is TMD active $>$
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Should TMD steering forces be applied to the system. If TMD is enabled, `TMDk`, `TMDFile`, and `TMDLastStep` must be defined in the input file as well.

- **TMDk** $<$ Elastic constant for TMD forces $>$
  Acceptable Values: Positive value in $kcal/mol/\text{Å}^2$.
  Description: The value of k in Eq. 9. A value of 200 seems to work well in many cases.

- **TMDOutputFreq** $<$ How often to print TMD output $>$
  Acceptable Values: Positive integer
  Default Value: 1
  Description: TMD output consists of lines of the form `TMD ts targetRMS currentRMS` where `ts` is the timestep, `targetRMS` is the target RMSD at that timestep, and `currentRMS` is the actual RMSD.

- **TMDFile** $<$ File for TMD information $>$
  Acceptable Values: Path to PDB file
  Description: Target atoms are those whose occupancy (O) is nonzero in the TMD PDB file. The file must contain the same number of atoms as the structure file. The coordinates for the target structure are also taken from the targeted atoms in this file. Non-targeted atoms are ignored.

- **TMDFirstStep** $<$ first TMD timestep $>$
  Acceptable Values: Positive integer
  Default Value: 0
  Description:

- **TMDLastStep** $<$ last TMD timestep $>$
  Acceptable Values: Positive integer
  Description: TMD forces are applied only between `TMDFirstStep` and `TMDLastStep`. The target RMSD evolves linearly in time from the initial to the final target value.

- **TMDInitialRMSD** $<$ target RMSD at first TMD step $>$
  Acceptable Values: Non-negative value in Å
  Default Value: from coordinates
  Description: In order to perform TMD calculations that involve restarting a previous NAMD run, be sure to specify `TMDInitialRMSD` with the same value in each NAMD input file, and use the NAMD parameter `firstTimestep` in the continuation runs so that the target RMSD continues from where the last run left off.

- **TMDFinalRMSD** $<$ target RMSD at last TMD step $>$
  Acceptable Values: Non-negative value in Å
  Default Value: 0
  Description: If no `TMDInitialRMSD` is given, the initial RMSD will be calculated at the

first TMD step. `TMDFinalRMSD` may be less than or greater than `TMDInitialRMSD`, depending on whether the system is to be steered towards or away from a target structure, respectively. Forces are applied only if $RMS(t)$ is betwween `TMDInitialRMSD` and $RMS * (t)$; in other words, only if the current RMSD fails to keep pace with the target value.

8.6 Steered Molecular Dynamics (SMD)

The SMD feature is independent from the harmonic constraints, although it follows the same ideas. In both SMD and harmonic constraints, one specifies a PDB file which indicates which atoms are 'tagged' as constrained. The PDB file also gives initial coordinates for the constraint positions. One also specifies such parameters as the force constant(s) for the constraints, and the velocity with which the constraints move.

There are two major differences between SMD and harmonic constraints:

- In harmonic constraints, each tagged atom is harmonically constrained to a reference point which moves with constant velocity. In SMD, it is the *center of mass* of the tagged atoms which is constrained to move with constant velocity.

- In harmonic constraints, each tagged atom is constrained in all three spatial dimensions. In SMD, tagged atoms are constrained *only along the constraint direction.*

The center of mass of the SMD atoms will be harmonically constrained with force constant k (`SMDk`) to move with velocity v (`SMDVel`) in the direction $\vec{n}$ (`SMDDir`). SMD thus results in the following potential being applied to the system:

$$U(\vec{r}_1, \vec{r}_2, ..., t) \;=\; \frac{1}{2}k \left[vt - (\vec{R}(t) - \vec{R}_0) \cdot \vec{n} \right]^2 . \tag{10}$$

Here, $t \equiv N_{ts}dt$ where N_{ts} is the number of elapsed timesteps in the simulation and dt is the size of the timestep in femtoseconds. Also, $\vec{R}(t)$ is the current center of mass of the SMD atoms and R_0 is the initial center of mass as defined by the coordinates in `SMDFile`. Vector $\vec{n}$ is normalized by NAMD before being used.

Output   NAMD provides output of the current SMD data. The frequency of output is specified by the `SMDOutputFreq` parameter in the configuration file. Every `SMDOutputFreq` timesteps NAMD will print the current timestep, current position of the center of mass of the restrained atoms, and the current force applied to the center of mass (in piconewtons, pN). The output line starts with word `SMD`

Parameters   The following parameters describe the parameters for the SMD feature of NAMD.

- `SMD` < Are SMD features active >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Should SMD harmonic constraint be applied to the system. If set to `on`, then `SMDk`, `SMDFile`, `SMDVel`, and `SMDDir` must be defined. Specifying `SMDOutputFreq` is optional.

- `SMDFile` < SMD constraint reference position >
  Acceptable Values: UNIX filename
  Description: File to use for the initial reference position for the SMD harmonic constraints.

All atoms in this PDB file with a nonzero value in the *occupancy* column will be tagged as SMD atoms. The coordinates of the tagged SMD atoms will be used to calculate the initial center of mass. During the simulation, this center of mass will move with velocity `SMDVel` in the direction `SMDDir`. The actual atom order in this PDB file must match that in the structure or coordinate file, since the atom number field in this PDB file will be ignored.

- `SMDk` < force constant to use in SMD simulation >
  Acceptable Values: positive real
  Description: SMD harmonic constraint force constant. Must be specified in kcal/mol/Å^2. The conversion factor is 1 kcal/mol = 69.479 pN Å.

- `SMDVel` < Velocity of the SMD reference position movement >
  Acceptable Values: nonzero real, Å/timestep
  Description: The velocity of the SMD center of mass movement. Gives the absolute value.

- `SMDDir` < Direction of the SMD center of mass movement >
  Acceptable Values: non-zero vector
  Description: The direction of the SMD reference position movement. The vector does not have to be normalized, it is normalized by NAMD before being used.

- `SMDOutputFreq` < frequency of SMD output >
  Acceptable Values: positive integer
  Default Value: 1
  Description: The frequency in timesteps with which the current SMD data values are printed out.

8.7 Interactive Molecular Dynamics (IMD)

NAMD now works directly with VMD to allow you to view and interactively steer your simulation. With IMD enabled, you can connect to NAMD at any time during the simulation to view the current state of the system or perform interactive steering.

- `IMDon` < is IMD active? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not to listen for an IMD connection.

- `IMDport` < port number to expect a connection on >
  Acceptable Values: positive integer
  Description: This is a free port number on the machine that node 0 is running on. This number will have to be entered into VMD.

- `IMDfreq` < timesteps between sending coordinates >
  Acceptable Values: positive integer
  Description: This allows coordinates to be sent less often, which may increase NAMD performance or be necessary due to a slow network.

- `IMDwait` < wait for an IMD connection? >
  Acceptable Values: yes or no
  Default Value: no

Description: If `no`, NAMD will proceed with calculations whether a connection is present or not. If `yes`, NAMD will pause at startup until a connection is made, and pause when the connection is lost.

- `IMDignore` < ignore interactive steering forces >
  Acceptable Values: `yes` or `no`
  Default Value: `no`
  Description: If `yes`, NAMD will ignore any steering forces generated by VMD to allow a simulation to be monitored without the possibility of perturbing it.

8.8 Tcl Forces and Analysis

NAMD provides a limited Tcl scripting interface designed for applying forces and performing on-the-fly analysis. This interface is efficient if only a few coordinates, either of individual atoms or centers of mass of groups of atoms, are needed. In addition, information must be requested one timestep in advance. To apply forces individually to a potentially large number of atoms, use `tclBC` instead as described in Sec. 8.9. The following configuration parameters are used to enable the Tcl interface:

- `tclForces` < is Tcl interface active? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether or not Tcl interface is active. If it is set to `off`, then no Tcl code is executed. If it is set to `on`, then Tcl code specified in `tclForcesScript` parameters is executed.

- `tclForcesScript` < input for Tcl interface >
  Acceptable Values: file or {script}
  Description: Must contain either the name of a Tcl script file or the script itself between { and } (may include multiple lines). This parameter may occur multiple times and scripts will be executed in order of appearance. The script(s) should perform any required initialization on the Tcl interpreter, including requesting data needed during the first timestep, and define a procedure `calcforces` { } to be called every timestep.

At this point only low-level commands are defined. In the future this list will be expanded. Current commands are:

- `print <anything>`
  This command should be used instead of `puts` to display output. For example, "`print Hello World`".

- `atomid <segname> <resid> <atomname>`
  Determines atomid of an atom from its segment, residue, and name. For example, "`atomid br 2 N`".

- `addatom <atomid>`
  Request coordinates of this atom for next force evaluation, and the calculated total force on this atom for current force evaluation. Request remains in effect until `clearconfig` is called. For example, "`addatom 4`" or "`addatom [atomid br 2 N]`".

78

- `addgroup <atomid list>`
  Request center of mass coordinates of this group for next force evaluation. Returns a group ID which is of the form `gN` where N is a small integer. This group ID may then be used to find coordinates and apply forces just like a regular atom ID. Aggregate forces may then be applied to the group as whole. Request remains in effect until `clearconfig` is called. For example, "`set groupid [addgroup { 14 10 12 }]`".

- `clearconfig`
  Clears the current list of requested atoms. After `clearconfig`, calls to `addatom` and `addgroup` can be used to build a new configuration.

- `getstep`
  Returns the current step number.

- `loadcoords <varname>`
  Loads requested atom and group coordinates (in Å) into a local array. `loadcoords` should only be called from within the `calcforces` procedure. For example, "`loadcoords p`" and "`print $p(4)`".

- `loadforces <varname>`
  Loads the forces applied in the previous timestep (in kcal mol^{-1} Å^{-1}) into a local array. `loadforces` should only be called from within the `calcforces` procedure. For example, "`loadforces f`" and "`print $f(4)`".

- `loadtotalforces <varname>`
  Loads the total forces on each requested atom in the previous time step (in kcal mol^{-1}Å^{-1}) into a local array. The total force also includes external forces. Note that the "`loadforces`" command returns external forces applied by the user. Therefore, one can subtract the external force on an atom from the total force on this atom to get the pure force arising from the simulation system.

- `loadmasses <varname>`
  Loads requested atom and group masses (in amu) into a local array. `loadmasses` should only be called from within the `calcforces` procedure. For example, "`loadcoords m`" and "`print $m(4)`".

- `addforce <atomid|groupid> <force vector>`
  Applies force (in kcal mol^{-1} Å^{-1}) to atom or group. `addforce` should only be called from within the `calcforces` procedure. For example, "`addforce $groupid { 1. 0. 2. }`".

- `addenergy <energy (kcal/mol)>`
  This command adds the specified energy to the MISC column (and hence the total energy) in the energy output. For normal runs, the command does not affect the simulation trajectory at all, and only has an artificial effect on its energy output. However, it can indeed affect minimizations.

With the commands above and the functionality of the Tcl language, one should be able to perform any on-the-fly analysis and manipulation. To make it easier to perform certain tasks, some Tcl routines are provided below.

Several vector routines (`vecadd`, `vecsub`, `vecscale`) from the VMD Tcl interface are defined. Please refer to VMD manual for their usage.

The following routines take atom coordinates as input, and return some geometry parameters (bond, angle, dihedral).

- `getbond <coor1> <coor2>`
  Returns the length of the bond between the two atoms. Actually the return value is simply the distance between the two coordinates. "coor1" and "coor2" are coordinates of the atoms.

- `getangle <coor1> <coor2> <coor3>`
  Returns the angle (from 0 to 180) defined by the three atoms. "coor1", "coor2" and "coor3" are coordinates of the atoms.

- `getdihedral <coor1> <coor2> <coor3> <coor4>`
  Returns the dihedral (from -180 to 180) defined by the four atoms. "coor1", "coor2", "coor3" and "coor4" are coordinates of the atoms.

The following routines calculate the derivatives (gradients) of some geometry parameters (angle, dihedral).

- `anglegrad <coor1> <coor2> <coor3>`
  An angle defined by three atoms is a function of their coordinates: $\theta\,(\vec{r_1}, \vec{r_2}, \vec{r_3})$ (in radian). This command takes the coordinates of the three atoms as input, and returns a list of $\{\frac{\partial\theta}{\partial\vec{r_1}}$ $\frac{\partial\theta}{\partial\vec{r_2}}$ $\frac{\partial\theta}{\partial\vec{r_3}}\}$. Each element of the list is a 3-D vector in the form of a Tcl list.

- `dihedralgrad <coor1> <coor2> <coor3> <coor4>`
  A dihedral defined by four atoms is a function of their coordinates: $\phi\,(\vec{r_1}, \vec{r_2}, \vec{r_3}, \vec{r_4})$ (in radian). This command takes the coordinates of the four atoms as input, and returns a list of $\{\frac{\partial\phi}{\partial\vec{r_1}}$ $\frac{\partial\phi}{\partial\vec{r_2}}$ $\frac{\partial\phi}{\partial\vec{r_3}}$ $\frac{\partial\phi}{\partial\vec{r_4}}\}$. Each element of the list is a 3-D vector in the form of a Tcl list.

As an example, here's a script which applies a harmonic constraint (reference position being 0) to a dihedral. Note that the "addenergy" line is not really necessary – it simply adds the calculated constraining energy to the MISC column, which is displayed in the energy output.

```
tclForcesScript {
  # The IDs of the four atoms defining the dihedral
  set aid1 112
  set aid2 123
  set aid3 117
  set aid4 115

  # The "spring constant" for the harmonic constraint
  set k 3.0

  addatom $aid1
  addatom $aid2
  addatom $aid3
  addatom $aid4
```

```
set PI 3.1416

proc calcforces {} {

  global aid1 aid2 aid3 aid4 k PI

  loadcoords p

  # Calculate the current dihedral
  set phi [getdihedral $p($aid1) $p($aid2) $p($aid3) $p($aid4)]
  # Change to radian
  set phi [expr $phi*$PI/180]

  # (optional) Add this constraining energy to "MISC" in the energy output
  addenergy [expr $k*$phi*$phi/2.0]

  # Calculate the "force" along the dihedral according to the harmonic constraint
  set force [expr -$k*$phi]

  # Calculate the gradients
  foreach {g1 g2 g3 g4} [dihedralgrad $p($aid1) $p($aid2) $p($aid3) $p($aid4)] {}

  # The force to be applied on each atom is proportional to its
  # corresponding gradient
  addforce $aid1 [vecscale $g1 $force]
  addforce $aid2 [vecscale $g2 $force]
  addforce $aid3 [vecscale $g3 $force]
  addforce $aid4 [vecscale $g4 $force]
  }
}
```

8.9   Tcl Boundary Forces

While the `tclForces` interface described above is very flexible, it is only efficient for applying forces to a small number of pre-selected atoms. Applying forces individually to a potentially large number of atoms, such as applying boundary conditions, is much more efficient with the `tclBC` facility described below.

- **tclBC** < are Tcl boundary forces active? >
  Acceptable Values: on or off
  Default Value: off
  Description: Specifies whether or not Tcl interface is active. If it is set to off, then no Tcl code is executed. If it is set to on, then Tcl code specified in the `tclBCScript` parameter is executed.

- **tclBCScript** < input for Tcl interface >
  Acceptable Values: {script}

Description: Must contain the script itself between { and } (may include multiple lines). This parameter may occur only once. The script(s) should perform any required initialization on the Tcl interpreter and define a procedure `calcforces <step> <unique> [args...]` to be called every timestep.

- `tclBCArgs` < extra args for tclBC calcforces command >
  Acceptable Values: {args...}
  Description: The string (or Tcl list) provided by this option is appended to the tclBC calcforces command arguments. This parameter may appear multiple times during a run in order to alter the parameters of the boundary potential function.

The script provided in `tclBCScript` and the `calcforces` procedure it defines are executed in multiple Tcl interpreters, one for every processor that owns patches. These `tclBC` interpreters do not share state with the Tcl interpreter used for `tclForces` or config file parsing. The `calcforces` procedure is passed as arguments the current timestep, a "unique" flag which is non-zero for exactly one Tcl interpreter in the simulation (that on the processor of patch zero), and any arguments provided to the most recent `tclBCArgs` option. The "unique" flag is useful to limit printing of messages, since the command is invoked on multiple processors.

The `print`, `vecadd`, `vecsub`, `vecscale`, `getbond`, `getangle`, `getdihedral`, `anglegrad`, and `dihedralgrad` commands described under `tclForces` are available at all times.

The `wrapmode <mode>` command, available in the `tclBCScript` or the `calcforces` procedure, determines how coordinates obtained in the calcforces procedure are wrapped around periodic boundaries. The options are:

- `patch`, (default) the position in NAMD's internal patch data structure, requires no extra calculation and is almost the same as `cell`

- `input`, the position corresponding to the input files of the simulation

- `cell`, the equivalent position in the unit cell centered on the `cellOrigin`

- `nearest`, the equivalent position nearest to the `cellOrigin`

The following commands are available from within the `calcforces` procedure:

- `nextatom`
  Sets the internal counter to a new atom and return 1, or return 0 if all atoms have been processed (this may even happen the first call). This should be called as the condition of a while loop, i.e., `while {[nextatom]} { ... }` to iterate over all atoms. One one atom may be accessed at a time.

- `dropatom`
  Excludes the current atom from future iterations on this processor until `cleardrops` is called. Use this to eliminate extra work when an atom will not be needed for future force calculations. If the atom migrates to another processor it may reappear, so this call should be used only as an optimization.

- `cleardrops`
  All available atoms will be iterated over by `nextatom` as if `dropatom` had never been called.

- **getcoord**
  Returns a list {x y z} of the position of the current atom wrapped in the periodic cell (if there is one) in the current wrapping mode as specified by `wrapmode`.

- **getcell**
  Returns a list of 1–4 vectors containing the cell origin (center) and as many basis vectors as exist, i.e., `{{ox oy oz} {ax ay az} {bx by bz} {cx cy cz}}`. It is more efficient to set the wrapping mode than to do periodic image calculations in Tcl.

- **getmass**
  Returns the mass of the current atom.

- **getcharge**
  Returns the charge of the current atom.

- **getid**
  Returns the 1-based ID of the current atom.

- **addforce {<fx> <fy> <fz>}**
  Adds the specified force to the current atom for this step.

- **addenergy <energy>**
  Adds potential energy to the BOUNDARY column of NAMD output.

As an example, these spherical boundary condition forces:

```
sphericalBC         on
sphericalBCcenter   0.0,0.0,0.0
sphericalBCr1       48
sphericalBCk1       10
sphericalBCexp1     2
```

Are replicated in the following script:

```
tclBC on
tclBCScript {
  proc veclen2 {v1} {
    foreach {x1 y1 z1} $v1 { break }
    return [expr $x1*$x1 + $y1*$y1 + $z1*$z1]
  }

  # wrapmode input
  # wrapmode cell
  # wrapmode nearest
  # wrapmode patch ;# the default

  proc calcforces {step unique R K} {
    if { $step % 20 == 0 } {
      cleardrops
```

```
    # if $unique { print "clearing dropped atom list at step $step" }
  }
  set R [expr 1.*$R]
  set R2 [expr $R*$R]
  set tol 2.0
  set cut2 [expr ($R-$tol)*($R-$tol)]

  while {[nextatom]} {
    # addenergy 1 ; # monitor how many atoms are checked
    set rvec [getcoord]
    set r2 [veclen2 $rvec]
    if { $r2 < $cut2 } {
      dropatom
      continue
    }
    if { $r2 > $R2 } {
      # addenergy 1 ; # monitor how many atoms are affected
      set r [expr sqrt($r2)]
      addenergy [expr $K*($r - $R)*($r - $R)]
      addforce [vecscale $rvec [expr -2.*$K*($r-$R)/$r]]
    }
  }
 }
}

tclBCArgs {48.0 10.0}
```

8.10 External Program Forces

This feature allows an external program to be called to calculate forces at every force evaluation, taking all atom coordinates as input.

- extForces < Apply external program forces? >
  Acceptable Values: yes or no
  Default Value: no
  Description: Specifies whether or not external program forces are applied.

- extForcesCommand < Force calculation command >
  Acceptable Values: UNIX shell command
  Description: This string is the argument to the system() function at every forces evaluation and should read coordinates from the file specified by extCoordFilename and write forces to the file specified by extForceFilename.

- extCoordFilename < Temporary coordinate file >
  Acceptable Values: UNIX filename
  Description: Atom coordinates are written to this file, which should be read by the extForcesCommand. The format is one line of "atomid charge x y z" for every atom followed by three lines with the periodic cell basis vectors "a.x a.y a.z", "b.x b.y b.z", and "c.x c.y

84

c.z". The atomid starts at 1 (not 0). For best performance the file should be in /tmp and not on a network-mounted filesystem.

- **extForceFilename** < Temporary force file >
  Acceptable Values: UNIX filename
  Description: Atom forces are read from this file after extForcesCommand in run. The format is one line of "atomid replace fx fy fz" for every atom followed by the energy on a line by itself and then, optionally, three lines of the virial "v.xx v.xy v.xz", "v.yx v.yy v.yz", "v.zx v.zy v.zz" where, e.g., v.xy = - fx * y for a non-periodic force. The atomid starts at 1 (not 0) and all atoms must be present and in order. The energy is added to the MISC output field. The replace flag should be 1 if the external program force should replace the forces calculated by NAMD for that atom and 0 if the forces should be added. For best performance the file should be in /tmp and not on a network-mounted filesystem.

9 Collective Variable-based Calculations[1]

The features described in this section were contributed by Giacomo Fiorin and Jérôme Hénin, Center for Molecular Modeling, University of Pennsylvania.

The collective variables module replaces and greatly extends the functionality of the older `freeEnergy` module for free energy of conformational change calculations. The older module is still available in the code, but has been removed from the documentation to discourage future use.

Today's molecular dynamics simulations involve a large number of degrees of freedom (typically 10,000–1,000,000), and it is often useful to reduce the description to a few parameters which can be either analyzed individually, or manipulated in order to alter the dynamics of the system in a controlled manner. Such reduced degrees of freedom have been called 'order parameters', 'collective variables', '(surrogate) reaction coordinates', and many other terms. In this section, the term 'collective variable' (shortened to *colvar*) is used, and it indicates any differentiable function of atomic Cartesian coordinates, $\boldsymbol{x}_i$, with i between 1 and N, the total number of atoms:

$$\xi(t) \ = \xi\left(\boldsymbol{x}_i(t), \boldsymbol{x}_j(t), \boldsymbol{x}_k(t), \ldots\right) \ , \ \ 1 \le i, j, k \ldots \le N \tag{11}$$

The collective variables module in NAMD offers a mechanism to define several collective variables, and execute various tasks on them. Its most important features are:

- defining an arbitrary number of colvars, and perform a multidimensional analysis or biased simulation by accessing any subset of colvars independently from the others (see 9.1);

- combine different functions of Cartesian coordinates (herein termed colvar *components*) into a colvar defined as a polynomial of several such components, thereby implementing new functional forms at runtime; periodic, multidimensional and symmetric components are handled transparently (see 9.2.1);

- calculate potentials of mean force (PMFs) or free energy surfaces for any set of colvars, using different sampling methods: currently implemented are the Adaptive Biasing Force (ABF) method (see 9.3.1), metadynamics (see 9.3.2), and Steered Molecular Dynamics (SMD) (see 9.3.3); any number of harmonic restraints can also be applied to the colvars (see 9.3.3);

- calculate statistical properties of an individual collective variable along a simulated trajectory, such as its running average, standard deviation or the time auto–correlation function (ACF) of its value, its velocity, or its total force (see 9.2.4);

- perform multidimensional correlation analysis, by calculating frequency histograms of any subset of colvars.

9.1 General parameters and input/output files

The structure of a typical collective variables calculation is depicted in Figure 5. Each colvar is a combination of one or more *components* (see 9.2), which are functions of several atomic coordinates. Multiple biasing or analysis methods can be applied to the same colvars. But care should be taken that certain methods (such as free energy reconstruction) do not function properly if other biases are adding forces to their colvars.

---

[1]The features described in this section were contributed by Giacomo Fiorin and Jérôme Hénin, Center for Molecular Modeling, University of Pennsylvania, Philadelphia, PA USA.
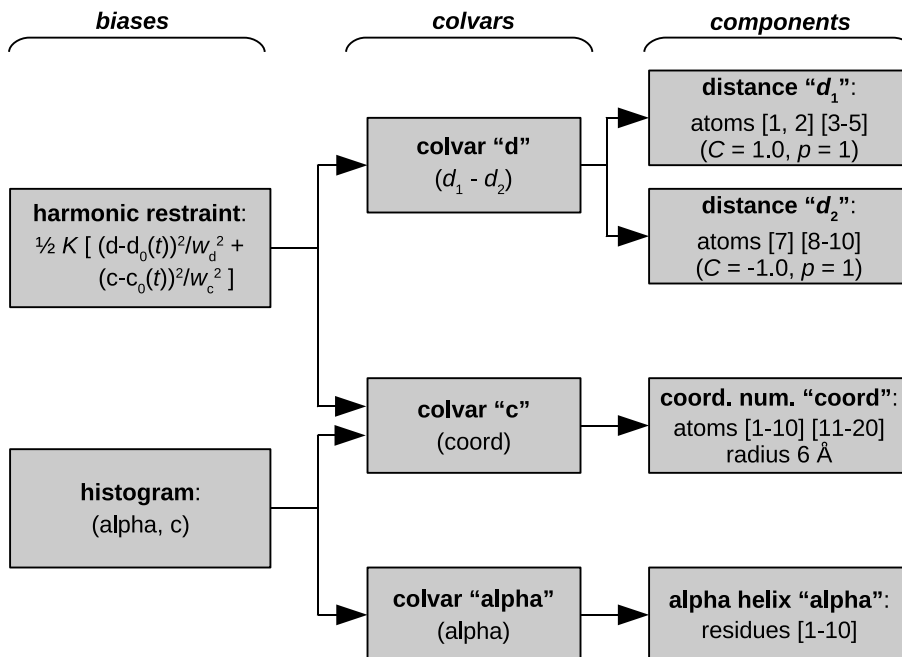
biases **colvars** **components**

harmonic restraint:
$\frac{1}{2} K [(d-d_0(t))^2/w_d^2 + (c-c_0(t))^2/w_c^2]$

histogram:
(alpha, c)

colvar "d"
$(d_1 - d_2)$

colvar "c"
(coord)

colvar "alpha"
(alpha)

distance "d_1":
atoms [1, 2] [3-5]
($C = 1.0$, $p = 1$)

distance "d_2":
atoms [7] [8-10]
($C = -1.0$, $p = 1$)

coord. num. "coord":
atoms [1-10] [11-20]
radius 6 Å

alpha helix "alpha":
residues [1-10]

Figure 5: Example of a collective variables (colvar) configuration. The colvar "d" is defined as the difference between two distances, each calculated between the centers of mass of two atom groups. The second colvar "c" holds the coordination number (i.e. the number of contacts) within a radius of 6 Å between two groups. The third colvar "alpha" measures the degree of α-helicity of the protein segment between residues 1 and 10. A moving harmonic restraint is applied to the colvars "d" and "c", each rescaled by means of width parameters w_d and w_c; the centers of the restraint, d_0 and c_0, evolve with the simulation time t. The histogram of "alpha" and "c" is also recorded on-the-fly.

9.1.1 NAMD parameters

To enable a colvar calculation, the NAMD configuration file must set two parameters (three when restarting a previous run):

- `colvars` < Enable the collective variables module >
  Acceptable Values: boolean
  Default Value: `off`
  Description: If this flag is on, the collective variables module within NAMD is executed at each time step; the module requires a separate configuration file, to be provided with `colvarsConfig`.

- `colvarsConfig` < Configuration file for the collective variables >
  Acceptable Values: UNIX filename
  Description: This file contains the definition of all collective variables and their biasing or analysis methods. It is meant to contain all the information that the user provides at the beginning of a simulation. In the case that additional information is needed to continue a simulation run, such information is read from the file provided by `colvarsInput`.

- `colvarsInput` < Input state file for the collective variables >
  Acceptable Values: UNIX filename
  Description: When continuing a previous simulation run, this file contains the information regarding the current state of all collective variables and their biasing methods. The format of this file is similar to that of `colvarsConfig`, but the set of keywords and their meaning are different. In normal circumstances, this file is written automatically at the end of a previous simulation, and the user does not need to modify it.

Othe parameters defining a collective variables calculation are not read from the NAMD configuration file, but from a separate file. As a consequence, keywords described in the following sections are not NAMD keywords.

9.1.2 Output files

By default, the collective variables module writes three output files:

- a *state file*, named <outputName>.colvars.state; this file is in ASCII format, regardless of the value of `binaryOutput` in the NAMD configuration; the name of this file can be provided as `colvarsInput` to continue the simulation in the next run;

- if either `colvarsRestartFrequency` or the NAMD parameter `restartFreq` is defined, a *restart file* (equivalent to the state file) is written every `colvarsRestartFrequency` steps, named <restartName>.colvars.state; the name of this file can be provided as `colvarsInput` to continue an interrupted run; (provided that the corresponding restart files for coordinates and velocities are used);

- if the colvars module parameter `colvarsTrajFrequency` is greater than 0 (default: 1), a *trajectory file* is written during the simulation, with the name <outputName>.colvars.traj; it is not needed to restart a simulation, but it can be read in for post–processing (see 9.2.4).

Other output files can written by specific methods applied to the colvars (e.g. by the ABF method, see 9.3.1, or the metadynamics method, see 9.3.2): as for the colvar trajectory file, they are meant for analyzing rather than continuing a simulation. All such files begin with the prefix <outputName>.

9.1.3 Colvars module configuration file

Figure 6 shows a typical collective variables configuration file. Its syntax is similar to that of the NAMD configuration file (2.2.1), with a few differences:

- the '=' sign between a keyword and its value, deprecated in the NAMD config, is not allowed in the colvars config (i.e. only 'keyword value' is valid, and not 'keyword = value');

- Tcl commands and syntax are not allowed;

- for keywords with a boolean value, in addition to the values yes|on|true and no|off|false, the keyword without any value evaluates as 'on'.

Options like `globalOption` define general features of the colvars module, and are described in the following. The options for the `colvar` blocks are documented in 9.2, and those for the colvar biases' blocks `biastype1`, `biastype2`, ... in 9.3.

```
# collective variables config file
globalOption value

colvar {
  name 1st-colvar # needed to identify the variable
  # flags to define the behavior of 1st-colvar
  colvarOption1 value1
  colvarOption2 value2
  # all the following terms are summed together to provide the
  # value of 1st-colvar
  componenttype1 {
    # configuration for the 1st colvar component
    componentOption value
    componentLongOption {
      ...
    }
  }
  componenttype2 {
    ...
  }
  ...
}

colvar {
  name 2nd-colvar
  ...
}

biastype1 {
  name mybias1
  colvars 1st-colvar 2nd-colvar
  ...
}

biastype2 { ... }
```

Figure 6: A typical colvars configuration file

- colvarsTrajFrequency < (global) Colvar value trajectory frequency >
  Acceptable Values: positive integer
  Default Value: 1
  Description: The values of each colvar (and any additional quantities which have been set to be reported) are written at this frequency to the file `<outputName>.colvars.traj`. If the value is 0, the trajectory file is not written. For optimization, the output is buffered (as is the NAMD log output in most operating systems), but it is synchronized with the disks every time the restart file is written.

- trajAppend < (global) Append to trajectory file >
  Acceptable Values: boolean
  Default Value: off
  Description: If this flag is enabled, and a file with the same name as the trajectory file is already present, new data is appended to that file. Otherwise, a new file is created.

- `colvarsRestartFrequency` < (global) Colvar module restart frequency >
  Acceptable Values: positive integer
  Default Value: `restartFreq`
  Description: Allows to choose a different restart frequency for the collective variables module. Redefining it may be useful to trace the time evolution of those few properties which are not written to the trajectory file for reasons of disk space.

- `analysis` < (global) Turn on analysis calculations >
  Acceptable Values: boolean
  Default Value: `off`
  Description: If this flag is enabled, the analysis mode is turned on, and all the colvars and biasing methods are instructed to perform whatever analysis they are configured to.

- `readTrajectory` < (global, analysis mode) Trajectory file to read >
  Acceptable Values: UNIX filename
  Description: If `analysis` is on, and the colvar module has been compiled as standalone, the values of the colvars is read from this file, as if a regular simulation was going on. The same configuration as that of the run which produced the trajectory should be used to ensure that data is read properly.

- `readBegin` < (global, analysis mode) Only read frames from this step >
  Acceptable Values: positive integer
  Default Value: `0`
  Description: If `readTrajectory` is defined, only lines starting from this step number are actually loaded by the colvars.

- `readEnd` < (global, analysis mode) Only read frames up to this step >
  Acceptable Values: positive integer
  Default Value: `0`
  Description: If `readTrajectory` is defined, and this number is larger than `readBegin`, only lines up to and including this step number are actually loaded in the colvars.

9.2 Declaring and using collective variables

Each collective variable is defined as a combination of individual quantities called *components* (see Figure 5). To obtain a colvar $\xi(\mathbf{r})$, its components $q_i(\mathbf{r})$ are summed according to:

$$\xi(\mathbf{r}) = \sum_i c_i [q_i(\mathbf{r})]^{n_i} \tag{12}$$

where each component appears with a unique coefficient c_i and (positive integer) exponent n_i. **Note:** *in most applications, only one component per colvar needs to be defined: accordingly, c_i and n_i are equal to 1 by default.*

In the configuration file, a collective variable is initiated by the keyword `colvar`, followed by its configuration options, usually between curly braces: `colvar {...}`. Each variable must have one or more components, defined by blocks specifying a functional form, atoms whose coordinates are used to evaluate this functional form, and optional parameters as needed. Each component block is initiated within the `colvar` block by the keyword corresponding to a given functional form. For

instance, the block `distance` {...} adds a component whose value is the distance between to atom groups.

The types of components used determine the properties of a `colvar`, and which biases or analysis methods can be applied to it. For instance, the `colvar` returns a scalar value if it includes components of one or more of the following types: `distance`, `distanceZ`, `distanceXY`, `angle`, `dihedral`, `coordnum`, `hBond`, `rmsd`, `orientationAngle`, `gyration`, and `alphaHelix`. The `colvar` returns instead a three–dimensional vector $\mathbf{d}$ if `distanceVec` is used, a three–dimensional vector with a $|\mathbf{d}|^2 = 1$ constraint if `distanceDir` is used, and a unitary quaternion (used to parameterize a rotation) if `orientation` is are used.

Currently, binning of colvar values on a grid can only be performed for scalar components. In addition, some properties like the system force, can be calculated only for some of these components. The detail documentation all component types is provided in 9.2.1. *Note: restrictions only apply to the type of an individual colvar. In fact, all of the implemented methods can be applied to any number of collective variables.*

Irrespective of which components it is built of, each `colvar` can be provided the following parameters:

- `name` < (`colvar`) Name of this colvar >
  Acceptable Values: string
  Default Value: "`colvar`" + numeric id
  Description: The name is an unique case–sensitive string which allows the colvar module to identify this `colvar` unambiguously; it is also used in the trajectory file to label to the columns corresponding to this `colvar`.

- `width` < (`colvar`) Typical fluctuation amplitude (or grid spacing) >
  Acceptable Values: positive decimal
  Default Value: 1.0
  Description: This number is a user–provided estimate of the typical fluctuation amplitude for this collective variable, or conversely, the typical width of a local free energy basin. Typically, twice the standard deviation of this variable during a short simulation run can be used. Biasing methods use this parameter for different purposes: harmonic restraints (9.3.3) use it to rescale the value of this colvar, ABF (9.3.1) interprets it as the grid spacing in the direction of this variable, and metadynamics (9.3.2) uses it as the width of newly added gaussians. This number is expressed in the same physical unit as the colvar value.

- `lowerBoundary` < (`colvar`) Lower boundary of the colvar >
  Acceptable Values: positive decimal
  Description: Defines the lowest possible value of this colvar that should be sampled in the simulation. It can either be a true physical boundary (under which the variable is not defined by construction), or an artificially imposed repulsive wall if `lowerWallConstant` is also defined. Together with `upperBoundary` and `width`, it defines a grid of values for the colvar. This number can only be defined for a scalar colvar.

- `upperBoundary` < (`colvar`) Upper boundary of the colvar >
  Acceptable Values: positive decimal
  Description: Similarly to `lowerBoundary`, defines the highest possible or allowed value.

- `lowerWallConstant` < (`colvar`) Lower boundary force constant (kcal/mol) >
  Acceptable Values: positive decimal

Description: If `lowerBoundary` is defined, enables a lower bounding restraint on the colvar in the form of a "half–harmonic" potential centered on `lowerBoundary`. The energy scale of the constant is kcal/mol, while the spatial scale is that of the `colvar`.

- `upperWallConstant` < (colvar) Upper boundary force constant (kcal/mol) >
  Acceptable Values: positive decimal
  Description: If `upperBoundary` is defined, enables an upper bounding restraint on the colvar. The unit is the same as `lowerWallConstant`.

- `outputValue` < (colvar) Output a trajectory for this `colvar` >
  Acceptable Values: boolean
  Default Value: on
  Description: If `colvarsTrajFrequency` is defined, the value of this `colvar` are written to the trajectory file every `colvarsTrajFrequency` steps in the column labeled "<name>".

- `outputVelocity` < (colvar) Output a velocity trajectory for this colvar >
  Acceptable Values: boolean
  Default Value: off
  Description: If `colvarsTrajFrequency` is defined, the finite–difference calculated velocity of this `colvar` are written to the trajectory file under the label "v_<name>".

- `outputSystemForce` < (colvar) Output a system force trajectory for this colvar >
  Acceptable Values: boolean
  Default Value: off
  Description: If `colvarsTrajFrequency` is defined, and all components support its calculation, the total system force on this `colvar` (i.e. the projection of all interatomic forces except constraint forces on this colvar — see equation (23) in section 9.3.1) are written to the trajectory file under the label "fs_<name>". The physical unit for this force is kcal/mol divided by the colvar unit.

- `outputAppliedForce` < (colvar) Output an applied force trajectory for this colvar >
  Acceptable Values: boolean
  Default Value: off
  Description: If `colvarsTrajFrequency` is defined, the total force applied on this `colvar` by biases within the colvar module are written to the trajectory under the label "fa_<name>". The physical unit for this force is kcal/mol divided by the colvar unit.

- `extendedLagrangian` < (colvar) Add an extra degree of freedom >
  Acceptable Values: boolean
  Default Value: off
  Description: Adds a fictitious particle with mass `fictitiousMass`, coupled to the colvar by a harmonic potential with force constant `extendedForceConstant`. Biasing forces on the colvar are applied to this extra particle, rather than to the atoms directly. This implements the extended Lagrangian formalism used in some metadynamics simulations [27].

- `extendedForceConstant` < (colvar) Ext. Lagrangian force constant (kcal/mol) >
  Acceptable Values: positive decimal
  Default Value: 1.0
  Description: Defines the force constant for the `extendedLagrangian` mode. The physical unit is the same as `lowerWallConstant`.

- **fictitiousMass** $<$ (colvar) Fictitious mass of the colvar (amu) $>$
  Acceptable Values: positive decimal
  Default Value: 1.0
  Description: Sets the fictitious inertial mass of this `colvar`.

9.2.1 Collective variable components

Each `colvar` has one or more *components*, each of them defined by a functional form, the atom positions from which it is calculated, and any additional parameters of the functional form. This section lists the types of components and their parameters. Each component makes use of one or more *atom groups*, whose syntax of definition is by their name followed by a definition block, e.g. `myatoms` {...}; the options in the definition block are described in (9.2.3).

Component `distance`: distance between two groups. The `distance` {...} block defines a distance component, between two atom groups, `group1` and `group2`.

- **group1** $<$ (distance) First group of atoms $>$
  Acceptable Values: Block `group1` {...}
  Description: First group of atoms.

- **group2** $<$ (distance) Second group of atoms $>$
  Acceptable Values: Block `group2` {...}
  Description: Second group of atoms.

- **oneSiteSystemForce** $<$ (distance) Measure system force on group 1 only? $>$
  Acceptable Values: boolean
  Default Value: no
  Description: If this is set to `yes`, the system force is measured along a vector field (see equation (23) in section 9.3.1) that only involves atoms of `group1`. This option is only useful for ABF, or custom biases that compute system forces. See section 9.3.1 for details.

The value returned is a positive number (in Å), limited between 0 and the largest possible interatomic distance within the chosen boundary conditions (in PBC, the minimum image convention is used).

Component `distanceZ`: projection of a distance on an axis. The `distanceZ` {...} block defines a distance projection component, which can be seen as measuring the distance between two groups projected onto an axis, or the position of a group along such an axis. The axis can be defined using either one reference group and a constant vector, or dynamically based on two reference groups.

- **main** $<$ (distanceZ, distanceXY) Main group of atoms $>$
  Acceptable Values: Block `main` {...}
  Description: Group of atoms whose position r is measured.

- **ref** $<$ (distanceZ, distanceXY) Reference group of atoms $>$
  Acceptable Values: Block `ref` {...}
  Description: Reference group of atoms. The position of its center of mass is noted r_1 below.

- ref2 < (distanceZ, distanceXY) Secondary reference group >
  Acceptable Values: Block ref2 {...}
  Default Value: none
  Description: Optional group of reference atoms, whose position r_2 can be used to define a dynamic projection axis: $e = (\|r_2 - r_1\|)^{-1} \times (r_2 - r_1)$. In this case, the origin is $r_m = 1/2(r_1 + r_2)$, and the value of the component is $e \cdot (r - r_m)$.

- axis < (distanceZ, distanceXY) Projection axis (Å) >
  Acceptable Values: (x, y, z) triplet
  Default Value: (0.0, 0.0, 1.0)
  Description: The three components of this vector define (when normalized) a projection axis e for the distance vector $r - r_1$ joining the centers of groups ref and main. The value of the component is then $e \cdot (r - r_1)$. The vector should be written as three components separated by commas and enclosed in parentheses.

- oneSiteSystemForce < (distanceZ, distanceXY) Measure system force on group main only? >
  Acceptable Values: boolean
  Default Value: no
  Description: If this is set to yes, the system force is measured along a vector field (see equation (23) in section 9.3.1) that only involves atoms of main. This option is only useful for ABF, or custom biases that compute system forces. See section 9.3.1 for details.

This component returns a number (in Å) whose range is determined by the chosen boundary conditions: if the z axis is used in PBC, its value is limited between $-b_z/2$ and $b_z/2$, where b_z is the box length in the z direction.

Component distanceXY: projection of a distance on a plane. The distanceXY {...} block defines a distance projected on a plane, and accepts the same options as distanceZ, i.e. main, ref, either ref2 or axis, and oneSiteSystemForce. It returns the norm of the projection of the distance vector between main and ref onto the plane orthogonal to the axis. The axis is defined using the axis parameter or as the vector joining ref and ref2 (see distanceZ above).

Component distanceVec: distance vector between two groups. The distanceVec {...} block defines a distance vector component, which accepts the same options as distance, group1 and group2. It returns a 3-dimensional vector $\mathbf{d} = (d_x, d_y, d_z)$. The range of values of d_x, d_y and d_z (in Å) is determined by the chosen boundary conditions.

Component distanceDir: distance unit vector between two groups. The distanceDir {...} block defines a distance unit vector component, which accepts the same options as distance, group1 and group2. It returns a 3-dimensional unit vector $\mathbf{d} = (d_x, d_y, d_z)$, $|\mathbf{d}| = 1$. d_x, d_y and d_z lie within the $[-1 : 1]$ interval.

Component angle: angle between three groups. The angle {...} block defines an angle, which accepts the options group1, group2 and group3 to define the three groups. It returns an angle (in degrees) within the interval $[0 : 180]$.

Component `dihedral`: torsional angle between four groups. The `dihedral {...}` block defines a torsional angle, which accepts the options `group1`, `group2`, `group3` and `group4` to define the four groups. It returns an angle (in degrees) within the interval $[-180 : 180]$. The colvar module calculates all the distances between two angles taking into account periodicity. For instance, reference values for restraints or range boundaries can be defined by using any real number of choice.

- `oneSiteSystemForce` $<$ (`dihedral`) Measure system force on group 1 only? $>$
  Acceptable Values: boolean
  Default Value: `no`
  Description: If this is set to `yes`, the system force is measured along a vector field (see equation (23) in section 9.3.1) that only involves atoms of `group1`. See section 9.3.1 for an example.

Component `coordnum`: coordination number between two groups. The `coordnum {...}` block defines a coordination number (or number of contacts), which calculates the function $(1 - (d/d_0)^n)/(1 - (d/d_0)^m)$, where d_0 is the "cutoff" distance, and n and m are exponents that can control its long range behavior and stiffness [27]. This function is summed over all pairs of atoms in `group1` and `group2`:

$$C(\texttt{group1}, \texttt{group2}) = \sum_{i \in \texttt{group1}} \sum_{j \in \texttt{group2}} \frac{1 - (|\mathbf{x}_i - \mathbf{x}_j|/d_0)^n}{1 - (|\mathbf{x}_i - \mathbf{x}_j|/d_0)^m} \tag{13}$$

This colvar component accepts the same options as `distance`, `group1` and `group2`. In addition to them, it recognizes the following options:

- `cutoff` $<$ (`coordnum`) Reference distance (Å) $>$
  Acceptable Values: positive decimal
  Default Value: 4.0
  Description: This number defines the switching distance to define an interatomic contact: for $d \ll d_0$, the switching function $(1 - (d/d_0)^n)/(1 - (d/d_0)^m)$ is close to 1, at $d = d_0$ it has a value of n/m (1/2 with the default n and m), and at $d \gg d_0$ it goes to zero approximately like d^{m-n}. Hence, for a proper behavior, m must be larger than n.

- `expNumer` $<$ (`coordnum`) Numerator exponent $>$
  Acceptable Values: positive even integer
  Default Value: 6
  Description: This number defines the n exponent for the switching function.

- `expDenom` $<$ (`coordnum`) Denominator exponent $>$
  Acceptable Values: positive even integer
  Default Value: 12
  Description: This number defines the m exponent for the switching function.

- `cutoff3` $<$ (`coordnum`) Reference distance vector (Å) $>$
  Acceptable Values: "(x, y, z)" triplet of positive decimals
  Default Value: (4.0, 4.0, 4.0)
  Description: The three components of this vector define three different cutoffs d_0 for each direction. This option is mutually exclusive with `cutoff`.

- **group2CenterOnly** $<$ (coordnum) Use only **group2**'s center of mass $>$
  Acceptable Values: boolean
  Default Value: off
  Description: If this option is on, only contacts between the atoms in **group1** and the center of mass of **group2** are calculated. By default, the sum extends over all pairs of atoms in **group1** and **group2**.

This component returns a dimensionless number, which ranges from approximately 0 (all inter-atomic distances much larger than the cutoff) to $N_{\mathrm{group1}} * N_{\mathrm{group2}}$ (all distances within the cutoff), or N_{group1} if **group2CenterOnly** is used. As a recommendation, at least one **group1** and **group2** should be of limited size (unless **group2CenterOnly** is used), because the size of the loop over all pairs grows with the product of the sizes of **group1** and **group2**.

Component hBond: hydrogen bond between two atoms. The **hBond** $\{\dots\}$ block defines a hydrogen bond, implemented as a coordination number (eq. 13) between the donor and the acceptor atoms. Therefore, it accepts the same options **cutoff** (with a different default value of 3.3 Å), **expNumer** (with a default value of 6) and **expDenom** (with a default value of 8). Unlike **coordnum**, it requires two atom numbers, **acceptor** and **donor**, to be defined. It returns an adimensional number, with values between 0 (acceptor and donor far outside the cutoff distance) and 1 (acceptor and donor much closer than the cutoff).

Component rmsd: root mean square displacement (RMSD) with respect to a reference structure. The block **rmsd** $\{\dots\}$ defines the root mean square replacement (RMSD) of a group of atoms with respect to a reference structure. For each set of coordinates $\{\mathbf{x}_1(t), \mathbf{x}_2(t), \dots \mathbf{x}_N(t)\}$, the colvar component **rmsd** calculates the optimal rotation $U^{\{\mathbf{x}_i(t)\} \to \{\mathbf{x}_i^{(\mathrm{ref})}\}}$ that best superimposes the coordinates $\{\mathbf{x}_i(t)\}$ onto a set of reference coordinates $\{\mathbf{x}_i^{(\mathrm{ref})}\}$. Both the current and the reference coordinates are centered on their centers of geometry, $\mathbf{x}_{\mathrm{cog}}(t)$ and $\mathbf{x}_{\mathrm{cog}}^{(\mathrm{ref})}$. The root mean square displacement is then defined as:

$$\mathrm{RMSD}(\{\mathbf{x}_i(t)\}, \{\mathbf{x}_i^{(\mathrm{ref})}\}) \; = \; \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left| U\left(\mathbf{x}_i(t) - \mathbf{x}_{\mathrm{cog}}(t)\right) - \left(\mathbf{x}_i^{(\mathrm{ref})} - \mathbf{x}_{\mathrm{cog}}^{(\mathrm{ref})}\right) \right|^2} \tag{14}$$

The optimal rotation $U^{\{\mathbf{x}_i(t)\} \to \{\mathbf{x}_i^{(\mathrm{ref})}\}}$ is calculated within the formalism developed in reference [15], which guarantees a continuous dependence of $U^{\{\mathbf{x}_i(t)\} \to \{\mathbf{x}_i^{(\mathrm{ref})}\}}$ with respect to $\{\mathbf{x}_i(t)\}$. The options for **rmsd** are:

- **atoms** $<$ (rmsd) Atom group $>$
  Acceptable Values: atoms $\{\dots\}$ block
  Description: Defines the group of atoms of which the RMSD should be calculated.

- **refPositions** $<$ (rmsd) Reference coordinates $>$
  Acceptable Values: space–separated list of (x, y, x) triplets
  Description: This option (mutually exclusive with **refPositionsFile** sets the reference coordinates to be compared with. The list should be as long as the atom group **atoms**. This option is independent from that with the same keyword within the **atoms** $\{\dots\}$ block.

- **refPositionsFile** < (**rmsd**) Reference coordinates file >
  Acceptable Values: UNIX filename
  Description: This option (mutually exclusive with **refPositions** sets the PDB file name for the reference coordinates to be compared with. The format of the PDB file is the same as that provided by **refPositionsFile** within the **atoms** {...} block, but is independent from that with the same keyword within the **atoms** {...} block.

- **refPositionsCol** < (**rmsd**) PDB column to use >
  Acceptable Values: X, Y, Z, O or B
  Default Value: O
  Description: If **refPositionsFile** is defined, and the file contains all the atoms in the topology, this option sets which PDB field is used to select the reference coordinates for **atoms**. Otherwise, this field is ignored.

- **refPositionsColValue** < (**rmsd**) Value in the PDB column >
  Acceptable Values: positive decimal
  Description: If defined, this value identifies in the PDB column **refPositionsCol** of the file **refPositionsFile** which atom positions are to be read. Otherwise, all positions with a non-zero value will be read.

This component returns a positive real number (in Å).

Component `orientation`: orientation with respect to a reference structure. The block `orientation {...}` returns the same optimal rotation used in the `rmsd` component to superimpose the coordinates $\{\mathbf{x}_i(t)\}$ onto a set of reference coordinates $\{\mathbf{x}_i^{(\text{ref})}\}$. It accepts all the options of `rmsd`: `atoms`, `refPositions`, `refPositionsFile` and `refPositionsCol`, and returns a four dimensional vector $\mathbf{q} = (q_0, q_1, q_2, q_3)$, with $\sum_i q_i^2 = 1$; this *quaternion* expresses the optimal rotation $\{\mathbf{x}_i(t)\} \rightarrow \{\mathbf{x}_i^{(\text{ref})}\}$ according to the formalism in reference [15]. The quaternion (q_0, q_1, q_2, q_3) can also be written as $(\cos(\theta/2), \sin(\theta/2)\mathbf{u})$, where θ is the angle and $\mathbf{u}$ the normalized axis of rotation; for example, a rotation of $90°$ around the z axis should be expressed as "(0.707, 0.0, 0.0, 0.707)". The script `quaternion2rmatrix.tcl` provides Tcl functions for converting to and from a 4×4 rotation matrix in a format suitable for usage in VMD.

  Hint: to restrain the rotation of a macromolecule in solution (e.g. when one of the cell dimensions is much longer than the others), it is possible to define a `colvar` with an `orientation` component, and restrain it throuh the `harmonic` bias around the "zero" rotation, (1.0, 0.0, 0.0, 0.0).

Component `orientationAngle`: angle of rotation with respect to a reference structure. The block `orientationAngle {...}` accepts the same options of `rmsd` and `orientation` (`atoms`, `refPositions`, `refPositionsFile` and `refPositionsCol`), but it returns instead the angle of rotation θ between the current and the reference positions. This angle is expressed in degrees within the range $[-180°:180°]$.

Component `gyration`: radius of gyration. The block `gyration {...}` defines the parameters for calculating the radius of gyration of a group of atomic positions $\{\mathbf{x}_1(t), \mathbf{x}_2(t), \dots \mathbf{x}_N(t)\}$ with

respect to their center of geometry, $\mathbf{x}_{\mathrm{cog}}(t)$:

$$R_{\mathrm{gyr}} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} |\mathbf{x}_i(t) - \mathbf{x}_{\mathrm{cog}}(t)|^2} \tag{15}$$

This component must contain one `atoms {...}` block to define the atom group, and returns a positive number, expressed in Å.

Component `alpha`: α-helix content of a protein segment. The block `alpha {...}` defines the parameters to calculate the helical content of a segment of protein residues. The α-helical content across the $N+1$ residues N_0 to $N_0 + N$ is calculated by the formula:

$$\alpha \left(C_\alpha^{(N_0)}, O^{(N_0)}, C_\alpha^{(N_0+1)}, O^{(N_0+1)}, \ldots N^{(N_0+5)}, C_\alpha^{(N_0+5)}, O^{(N_0+5)}, \ldots N^{(N_0+N)}, C_\alpha^{(N_0+N)} \right) = \tag{16}$$

$$\frac{1}{2(N-2)} \sum_{n=N_0}^{N_0+N-2} \mathrm{angf}\left(C_\alpha^{(n)}, C_\alpha^{(n+1)}, C_\alpha^{(n+2)} \right) + \frac{1}{2(N-4)} \sum_{n=N_0}^{N_0+N-4} \mathrm{hbf}\left(O^{(n)}, N^{(n+4)} \right) \tag{17}$$

where the score function for the $C_\alpha - C_\alpha - C_\alpha$ angle is defined as:

$$\mathrm{angf}\left(C_\alpha^{(n)}, C_\alpha^{(n+1)}, C_\alpha^{(n+2)} \right) = \frac{1 - \left(\theta(C_\alpha^{(n)}, C_\alpha^{(n+1)}, C_\alpha^{(n+2)}) - \theta_0 \right)^2 / (\Delta\theta_{\mathrm{tol}})^2}{1 - \left(\theta(C_\alpha^{(n)}, C_\alpha^{(n+1)}, C_\alpha^{(n+2)}) - \theta_0 \right)^4 / (\Delta\theta_{\mathrm{tol}})^4} \tag{18}$$

and the score function for the $O^{(n)} \leftrightarrow N^{(n+4)}$ hydrogen bond is defined through a `hBond` colvar component on the same atoms. The options recognized within the `alpha {...}` block are:

- `residueRange` $<$ (`alpha`) Potential α-helical residues $>$
  Acceptable Values: "<Initial residue number>-<Final residue number>"
  Description: This option specifies the range of residues on which this component should be defined. The colvar module looks for the atoms within these residues named "`CA`", "`N`" and "`O`", and raises an error if any of those atoms is not found.

- `psfSegID` $<$ (`alpha`) PSF segment identifier $>$
  Acceptable Values: string (max 4 characters)
  Description: This option sets the PSF segment identifier for the residues specified in `residueRange`. This option need not be provided when non–PSF topologies are used by NAMD.

- `hBondCoeff` $<$ (`alpha`) Coefficient for the hydrogen bond term $>$
  Acceptable Values: positive between 0 and 1
  Default Value: 0.5
  Description: This number specifies the contribution to the total value from the hydrogen bond terms. 0 will disable the hydrogen bond terms, 1 will disable the angle terms.

- `angleRef` $<$ (`alpha`) Reference $C_\alpha - C_\alpha - C_\alpha$ angle $>$
  Acceptable Values: positive decimal
  Default Value: $88°$
  Description: This option sets the reference angle used in the score function (18).

- **angleTol** < (`alpha`) Tolerance in the $C_\alpha - C_\alpha - C_\alpha$ angle >
  Acceptable Values: positive decimal
  Default Value: $15°$
  Description: This option sets the angle tolerance used in the score function (18).

- **hBondCutoff** < (`alpha`) Hydrogen bond cutoff >
  Acceptable Values: positive decimal
  Default Value: 3.3 Å
  Description: Equivalent to the `cutoff` option in the `hBond` component.

- **hBondExpNumer** < (`alpha`) Hydrogen bond numerator exponent >
  Acceptable Values: positive integer
  Default Value: 6
  Description: Equivalent to the `expNumer` option in the `hBond` component.

- **hBondExpDenom** < (`alpha`) Hydrogen bond denominator exponent >
  Acceptable Values: positive integer
  Default Value: 8
  Description: Equivalent to the `expDenom` option in the `hBond` component.

This component returns positive values, always comprised between 0 (lowest α-helical score) and 1 (highest α-helical score).

9.2.2 Linear and polynomial combinations of components

Any set of components can be combined within a colvar, provided that they return the same type of values (scalar, unit vector, vector, or quaternion). By default, the colvar is the sum of its components. Linear or polynomial combinations (following equation (12)) can be obtained by setting the following parameters, which are common to all components:

- **componentExp** < (any component) Exponent of this component in the colvar >
  Acceptable Values: integer
  Default Value: 1
  Description: Defines the power at which the value of this component is raised before being added to the sum. When this exponent is different than 1 (non–linear sum), system forces and the Jacobian force are not available, making the colvar unsuitable for ABF calculations.

- **componentCoeff** < (any component) Coefficient of this component in the colvar >
  Acceptable Values: decimal
  Default Value: 1.0
  Description: Defines the coefficient by which this component is multiplied (after being raised to `componentExp`) before being added to the sum.

9.2.3 Defining atom groups

Each component depends on one or more *atom groups*, which can be defined by different methods in the configuration file. Each atom group block is initiated by the name of the group itself within the component block, followed by the instructions to the colvar module on how to select the atoms involved. Here is an example configuration, for an atom group called `myatoms`, which makes use of the most common keywords:

```
# atom group definition
myatoms {
  # add atoms 1, 2 and 3 to this group (note: numbers start from 1)
  atomNumbers {
     1 2 3
  }
  # add all the C-alphas within residues 11 to 20 of segment "PROT"
  atomNameResidueRange  CA 11-20
  psfSegID              PROT
  # add all the atoms with occupancy 2 in the file atoms_occ1.pdb
  atomsFile             atoms_occ1.pdb
  atomsCol              O
  atomsColValue         2.0
}
```

For any atom group, the available options are:

- **atomNumbers** < (atom group) List of atom numbers >
  Acceptable Values: space–separated list of positive integers
  Description: This option adds to the group all the atoms whose numbers are in the list.
  Atom numbering starts from 1.

- **atomNumbersRange** < (atom group) Atoms within a number range >
  Acceptable Values: <Starting number>-<Ending number>
  Description: This option adds to the group all the atoms whose numbers are within the range specified. It can be used multiple times for the same group. Atom numbering starts from 1. May be repeated.

- **atomNameResidueRange** < (atom group) Named atoms within a range of residue numbers >
  Acceptable Values: <Atom name> <Starting residue>-<Ending residue>
  Description: This option adds to the group all the atoms with the provided name within residues in the range provided. May be repated.

- **psfSegID** < (atom group) PSF segment identifier >
  Acceptable Values: string (max 4 characters)
  Description: This option sets the PSF segment identifier for all instances of atomNameResidueRange within this atom group. This option need not be provided when non–PSF topologies are used by NAMD.

- **atomsFile** < (atom group) PDB file name for atom selection >
  Acceptable Values: string
  Description: This option selects atoms from the PDB file provided and adds them to the group according to the value in the column atomsCol. **Note:** the PDB file provide must match the number of atoms in the topology.

- **atomsCol** < (atom group) PDB column to use for the selection >
  Acceptable Values: X, Y, Z, O or B
  Default Value: O

Description: This option specifies which column in `atomsFile` is used to determine the atoms to be included in the group.

- `atomsColValue` < (atom group) Value in the PDB column >
  Acceptable Values: positive decimal
  Description: If defined, this value identifies in `atomsCol` of `atomsFile` which atoms are to be read; otherwise, all atoms with a non-zero value will be read.

- `dummyAtom` < (atom group) Dummy atom position (Å) >
  Acceptable Values: `(x, y, z)` triplet
  Description: If no actual atoms are defined for the group, this option may be used to make this group return a fixed position in space. This is useful e.g. to make colvar components that normally calculate functions of the group's center of mass work with an absolute reference position. If this option is specified for the group, `disableForces` is turned on, the `(x, y, z)` position provided is returned as its center of mass, and zero velocities and system forces are reported.

- `centerReference` < (atom group) Ignore the translations of this group >
  Acceptable Values: boolean
  Default Value: `off`
  Description: If this option is `on`, the center of mass of this group is centered on a reference frame, determined either by `refPositions` or `refPositionsFile`. This transformation occurs *before* any colvar component has access to the coordinates of the group: hence, only the recentered coordinates are available to the colvars.

- `rotateReference` < (atom group) Ignore the rotations of this group >
  Acceptable Values: boolean
  Default Value: `off`
  Description: If this option is `on`, this group is rotated around its center of mass to optimally reproduce the coordinates given by `refPositions` or `refPositionsFile`. This is done before recentering the group, if `centerReference` is also defined. The algorithm used is the same employed in the `orientation` colvar component [15]. Forces applied to this group from within the colvar module are rotated back to the reference frame of the simulation prior being communicated to the MD integrator. **Note**: care must be taken when a torque is applied on the group, which could bring the simulation out of control; to avoid such problems, `disableForces` could be used for this group.

- `refPositions` < (atom group) Reference positions for the group (Å) >
  Acceptable Values: space–separated list of `(x, y, z)` triplets
  Description: If either `centerReference` or `rotateReference` is `on`, these coordinates are used to determine the center of mass translation and the optimal rotation between them and the current coordinates of the group. In the latter case, the list must be of the same length as this atom group.

- `refPositionsFile` < (atom group) Reference positions for the group >
  Acceptable Values: UNIX filename
  Description: If either `centerReference` or `rotateReference` is `on`, the coordinates from this file are used to determine the center of mass's translation and the optimal rotation between them and the current coordinates of the group. This file can either *i)* contain as

many atoms as the group (in which case all of the `ATOM` records are read) or *ii)* a larger number of atoms (in which case atoms will be selected according to `refPositionsCol`). A typical usage may be to reuse `atomsFile`. Unlike `atomsFile`, however, it need not match the number of atoms in the topology.

- `refPositionsCol` < (atom group) Reference positions for the group >
  Acceptable Values: X, Y, Z, O or B
  Default Value: O
  Description: Analogous to `atomsCol`.

- `refPositionsColValue` < (atom group) Value in the PDB column >
  Acceptable Values: positive decimal
  Description: Analogous to `atomsColValue`.

- `refPositionsGroup` < (atom group) Group to be used >
  Acceptable Values: Block `refPositionsGroup { ... }`
  Default Value: This group itself
  Description: If either `centerReference` or `rotateReference` is defined, this keyword allows to define an additional atom group, which is used instead of this one to calculate the translation or the rotation to the reference positions.

- `disableForces` < (atom group) Disable application of colvar forces on this group >
  Acceptable Values: boolean
  Default Value: off
  Description: If this option is `on`, all the forces applied from the colvars to the atoms in this group are ignored. However, the applied forces on each colvar are still written to the trajectory file, if requested. In some cases it may be desirable to use this option in order not to perturb the motion of selected groups of atoms. However, it is not advisable to perform a potential of mean force (PMF) calculation with this flag, because the biasing forces would not be applied uniformly. Also, when disabling the forces on one of several groups defining a colvar, a non–zero net force (or torque) is applied to the entire system, leading to an undesired center of mass motion (resp. rotation).

To minimize the length of the NAMD standard output, messages from the previous options are not echoed by default. They can be enabled by using the boolean keyword `verboseOutput`.

Recommendations for using atom groups.   Nearly all the calculations of colvar components are based on user–specified atom groups. These guidelines could be useful when choosing the atom groups to define a collective variable:

- When using periodic boundary conditions, NAMD maintains the coordinates of all the atoms within a molecule contiguous to each other (i.e. there are no "jumps" due to the periodic boundary conditions). The colvar module relies on this when calculating a group's center of mass, but this condition may fail when the group spans different molecules. In this case, writing the NAMD output files `wrapAll` or `wrapWater` could produce wrong results when a simulation run is continued from a previous one. There are however cases in which `wrapAll` or `wrapWater` can be safely applied:

  i) the group has only one atom;

ii) it has all its atoms within the same molecule;

iii) it is used by a colvar component which does not access its center of mass (`coordnum`, `hBond`, `alpha`);

iv) it is used by a colvar component that ignores the ill–defined Cartesian components of its center of mass (such as the x and y components of a membrane's center of mass by `distanceZ`).

In the general case, the user should determine, according to which type of calculation is being performed, whether `wrapAll` or `wrapWater` can be enabled.

- The colvars calculation is is not parallelized over the computational nodes. Therefore, the extra load on the first node, where the colvars module is executed, should be maintained lower than that of any other node. In most cases this condition is very well satisfied, but there may be some specific configurations for which such extra load may affect the parallel performance of NAMD. As a general guideline, an atom group should include for a large fraction of the whole system only when necessary.

9.2.4 Statistical analysis of individual collective variables

When the global keyword `analysis` is defined in the configuration file, calculations of statistical properties for individual colvars can be performed. At the moment, several types of time correlation functions, running averages and running standard deviations are available.

- `corrFunc` < (colvar) Calculate a time correlation function? >
  Acceptable Values: boolean
  Default Value: `off`
  Description: Whether or not a time correlaction function should be calculated for this colvar.

- `corrFuncWithColvar` < (colvar) Colvar name for the correlation function >
  Acceptable Values: string
  Description: By default, the auto-correlation function (ACF) of this colvar, ξ_i, is calculated. When this option is specified, the correlation function is calculated instead with another colvar, ξ_j, which must be of the same type (scalar, vector, or quaternion) of ξ_i.

- `corrFuncType` < (colvar) Type of the correlation function >
  Acceptable Values: `velocity`, `coordinate` or `coordinate_p2`
  Default Value: `velocity`
  Description: With `coordinate` or `velocity`, the correlation function $C_{i,j}(t) = \langle O\left(\xi_i(t_0), \xi_j(t_0 + t)\right)\rangle$ is calculated between the variables ξ_i and ξ_j, or their velocities. $O(\xi_i, \xi_j)$ is the scalar product when calculated between scalar or vector values, whereas for quaternions it is the cosine between the two corresponding rotation axes. With `coordinate_p2`, the second order Legendre polynomial, $(3\cos(\theta)^2 - 1)/2$, is used instead of the cosine.

- `corrFuncNormalize` < (colvar) Normalize the time correlation function? >
  Acceptable Values: boolean
  Default Value: `on`
  Description: If enabled, the value of the correlation function at $t = 0$ is normalized to 1; otherwise, it equals to $\langle O\left(\xi_i, \xi_j\right)\rangle$.

- **corrFuncLength** < (colvar) Length of the time correlation function >
  Acceptable Values: positive integer
  Default Value: 1000
  Description: Length (in number of points) of the time correlation function.

- **corrFuncStride** < (colvar) Stride of the time correlation function >
  Acceptable Values: positive integer
  Default Value: 1
  Description: Number of steps between two values of the time correlation function.

- **corrFuncOffset** < (colvar) Offset of the time correlation function >
  Acceptable Values: positive integer
  Default Value: 0
  Description: The starting time (in number of steps) of the time correlation function (default: $t = 0$). Note: the value at $t = 0$ is always used for the normalization.

- **corrFuncOutputFile** < (colvar) Output file for the time correlation function >
  Acceptable Values: UNIX filename
  Default Value: <name>.corrfunc.dat
  Description: The time correlation function is saved in this file.

- **runAve** < (colvar) Calculate the running average and standard deviation >
  Acceptable Values: boolean
  Default Value: off
  Description: Whether or not the running average and standard deviation should be calculated for this colvar.

- **runAveLength** < (colvar) Length of the running average window >
  Acceptable Values: positive integer
  Default Value: 1000
  Description: Length (in number of points) of the running average window.

- **runAveStride** < (colvar) Stride of the running average window values >
  Acceptable Values: positive integer
  Default Value: 1
  Description: Number of steps between two values within the running average window.

- **runAveOutputFile** < (colvar) Output file for the running average and standard deviation >
  Acceptable Values: UNIX filename
  Default Value: <name>.runave.dat
  Description: The running average and standard deviation are saved in this file.

9.3 Biasing and analysis methods

All of the biasing and analysis methods implemented recognize the following options:

- **name** < (colvar bias) Identifier for the bias >
  Acceptable Values: string
  Default Value: <type of bias><bias index>

Description: This string is used to identify the bias or analysis method in output messages and to name some output files.

- `colvars` < (colvar bias) Collective variables involved >
  Acceptable Values: space–separated list of `colvar` names
  Description: This option selects by name all the colvars to which this bias or analysis will be applied.

9.3.1 Adaptive Biasing Force calculations

For a full description of the Adaptive Biasing Force method, see reference [17]. For details about this implementation, see reference [23].

ABF is based on the thermodynamic integration (TI) scheme for computing free energy profiles. The free energy as a function of a set of collective variables $\boldsymbol{\xi} = (\xi_i)_{i \in [1,n]}$ is defined from the canonical distribution of $\boldsymbol{\xi}$, $\mathcal{P}(\boldsymbol{\xi})$:

$$A(\boldsymbol{\xi}) = -\frac{1}{\beta} \ln \mathcal{P}(\boldsymbol{\xi}) + A_0 \tag{19}$$

In the TI formalism, the free energy is obtained from its gradient, which is generally calculated in the form of the average of a force $\boldsymbol{F}_\xi$ exerted on $\boldsymbol{\xi}$, taken over an iso-$\boldsymbol{\xi}$ surface:

$$\boldsymbol{\nabla}_\xi A(\boldsymbol{\xi}) = \langle -\boldsymbol{F}_\xi \rangle_{\boldsymbol{\xi}} \tag{20}$$

Several formulae that take the form of (20) have been proposed. This implementation relies partly on the classic formulation [11], and partly on a more versatile scheme originating in a work by Ruiz-Montero et al. [39], generalized by den Otter [18] and extended to multiple variables by Ciccotti et al. [14]. Consider a system subject to constraints of the form $\sigma_k(\boldsymbol{x}) = 0$. Let $(\boldsymbol{v}_i)_{i \in [1,n]}$ be arbitrarily chosen vector fields ($\mathbb{R}^{3N} \to \mathbb{R}^{3N}$) verifying, for all i, j, and k:

$$\boldsymbol{v}_i \cdot \boldsymbol{\nabla}_{\boldsymbol{x}} \xi_j = \delta_{ij} \tag{21}$$

$$\boldsymbol{v}_i \cdot \boldsymbol{\nabla}_{\boldsymbol{x}} \sigma_k = 0 \tag{22}$$

then the following holds [14]:

$$\frac{\partial A}{\partial \xi_i} = \langle \boldsymbol{v}_i \cdot \boldsymbol{\nabla}_{\boldsymbol{x}} V - k_B T \boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{v}_i \rangle_{\boldsymbol{\xi}} \tag{23}$$

where V is the potential energy function. $\boldsymbol{v}_i$ can be interpreted as the direction along which the force acting on variable ξ_i is measured, whereas the second term in the average corresponds to the geometric entropy contribution that appears as a Jacobian correction in the classic formalism [11]. Condition (21) states that the direction along which the system force on ξ_i is measured is orthogonal to the gradient of ξ_j, which means that the force measured on ξ_i does not act on ξ_j.

Equation (22) implies that constraint forces are orthogonal to the directions along which the free energy gradient is measured, so that the measurement is effectively performed on unconstrained degrees of freedom. In NAMD, constraints are typically applied to the lengths of bonds involving hydrogen atoms, for example in TIP3P water molecules (parameter `rigidBonds`, section 5.3.1).

In the framework of ABF, $\mathbf{F}_\xi$ is accumulated in bins of finite size, $\delta\xi$, thereby providing an estimate of the free energy gradient according to equation (20). The biasing force applied along the colective variables to overcome free energy barriers is calculated as:

$$\mathbf{F}^{\mathrm{ABF}} = \boldsymbol{\nabla}_{\boldsymbol{x}} \, \widetilde{A}(\boldsymbol{\xi}) \tag{24}$$

where $\boldsymbol{\nabla}_{\boldsymbol{x}} \, \widetilde{A}$ denotes the current estimate of the free energy gradient at the current point $\boldsymbol{\xi}$ in the collective variable subspace.

As sampling of the phase space proceeds, the estimate $\boldsymbol{\nabla}_{\boldsymbol{x}} \, \widetilde{A}$ is progressively refined. The biasing force introduced in the equations of motion guarantees that in the bin centered around $\boldsymbol{\xi}$, the forces acting along the selected collective variables average to zero over time. Eventually, as the undelying free energy surface is canceled by the adaptive bias, evolution of the system along $\boldsymbol{\xi}$ is governed mainly by diffusion. Although this implementation of ABF can in principle be used in arbitrary dimension, a higher-dimension collective variable space is likely to result in sampling difficulties. Most commonly, the number of variables is one or two.

ABF requirements on collective variables

1. *Only linear combinations* of colvar components can be used in ABF calculations.

2. *Availability of system forces* is necessary. The following colvar components can be used in ABF calculations: `distance`, `distance_xy`, `distance_z`, `dihedral`, `gyration`, and `rmsd`.

3. *Mutual orthogonality of colvars.* In a multidimensional ABF calculation, equation (21) must be satisfied for any two colvars ξ_i and ξ_j. Various cases fulfill this orthogonality condition:

   - ξ_i and ξ_j are based on non-overlapping sets of atoms.
   - atoms involved in the force measurement on ξ_i do not participate in the definition of ξ_j. This can be obtained using the option `oneSiteSystemForce` of the `distance` and `dihedral` components (example: Ramachandran angles ϕ, ψ).
   - ξ_i and ξ_j are orthogonal by construction. Useful cases are the sum and difference of two components, or `distance_z` and `distance_xy` using the same axis.

4. *Mutual orthogonality of components*: when several components are combined into a colvar, it is assumed that their vectors $\boldsymbol{v}_i$ (equation (23)) are mutually orthogonal. The cases described for colvars in the previous paragraph apply.

5. *Orthogonality of colvars and constraints*: equation 22 can be satisfied in two simple ways, if either no constrained atoms are involved in the force measurement (see point 3 above) or pairs of atoms joined by a constraint bond are part of an *atom group* which only intervenes through its center (center of mass or geometric center) in the force measurement. In the latter case, the contributions of the two atoms to the left-hand side of equation 22 cancel out. For example, all atoms of a rigid TIP3P water molecule can safely be included in an atom group used in a `distance` component.

Parameters for ABF

The following parameters can be set in the ABF configuration block (in addition to generic bias parameters such as `colvars`):

- `fullSamples` < (ABF) Number of samples in a bin prior to application of the ABF >
  Acceptable Values: positive integer

Default Value: 200

Description: To avoid nonequilibrium effects in the dynamics of the system, due to large fluctuations of the force exerted along the reaction coordinate, ξ, it is recommended to apply the biasing force only after a reasonable estimate of the latter has been obtained.

- `hideJacobian` < (ABF) Remove geometric entropy term from calculated free energy gradient? >

  Acceptable Values: boolean

  Default Value: `no`

  Description: In a few special cases, most notably distance-based variables, an alternate definition of the potential of mean force is traditionally used, which excludes the Jacobian term describing the effect of geometric entropy on the distribution of the variable. This results, for example, in particle-particle potentials of mean force being flat at large separations. Setting this parameter to `yes` causes the output data to follow that convention, by removing this contribution from the output gradients while applying internally the corresponding correction to ensure uniform sampling. It is not allowed for colvars with multiple components.

- `outputFreq` < (ABF) Frequency (in timesteps) at which ABF data files are refreshed >

  Acceptable Values: positive integer

  Default Value: Colvar module restart frequency

  Description: The files containing the free energy gradient estimate and the sampling histogram (and the PMF if the calculation is one-dimensional) are written on disk at the given time interval.

- `inputPrefix` < (ABF) Filename prefix for reading ABF data >

  Acceptable Values: list of strings

  Description: If this parameter is set, for each item in the list, ABF tries to read a gradient and a sampling files named <`inputPrefix`>.`grad` and <`inputPrefix`>.`count`. This is done at startup and sets the initial state of the ABF algorithm. The data from all provided files is combined appropriately. Also, the grid definition (min and max values, width) need not be the same that for the current run. This command is useful to piece together data from simulations in different regions of collective variable space, or change the colvar boundary values and widths. Note that it is not recommended to use it to switch to a smaller width, as that will leave some bins empty in the finer data grid. This option is NOT compatible with reading the data from a restart file (`colvarsInput` option of the NAMD config file).

- `applyBias` < (ABF) Apply the ABF bias? >

  Acceptable Values: boolean

  Default Value: `yes`

  Description: If this is set to no, the calculation proceeds normally but the adaptive biasing force is not applied. Data is still collected to compute the free energy gradient. This is mostly intended for testing purposes, and should not be used in routine simulations.

ABF also depends on parameters from collective variables to define the grid on which free energy gradients are computed. In the direction of each colvar, the grid ranges from `lowerBoundary` to `upperBoundary`, and the bin width (grid spacing) is set by the `width` parameter.

Output files

The ABF bias produces the following files, all in multicolumn ASCII format:

- <outputName>.grad: current estimate of the free energy gradient (grid), in multicolumn;

- <outputName>.count: total number of samples collected, on the same grid;

- <outputName>.pmf: only for one-dimensional calculations, integrated free energy profile or PMF.

If several ABF biases are defined concurrently, their name is inserted to produce unique filenames for output, as in <outputName>.abf1.grad. This should not be done routinely and could lead to meaningless results: only do it if you know what you are doing!

If the colvar space has been partitioned into sections (*windows*) in which independent ABF simulations have been run, the resulting data can be merged using the inputPrefix option described above (a NAMD run of 0 steps is enough).

Reconstructing a multidimensional free energy surface

If a one-dimensional calculation is performed, the estimated free energy gradient is automatically integrated and a potential of mean force is written under the file name <outputName>.pmf, in a plain text format that can be read by most data plotting and analysis programs (e.g. gnuplot).

In dimension 2 or greater, integrating the discretized gradient becomes non-trivial. The standalone utility abf_integrate is provided to perform that task. abf_integrate reads the gradient data and uses it to perform a Monte-Carlo (M-C) simulation in discretized collective variable space (specifically, on the same grid used by ABF to discretize the free energy gradient). By default, a history-dependent bias (similar in spirit to metadynamics) is used: at each M-C step, the bias at the current position is incremented by a preset amount (the *hill height*). Upon convergence, this bias counteracts optimally the underlying gradient; it is negated to obtain the estimate of the free energy surface.

abf_integrate is invoked using the command-line:

```
integrate <gradient_file> [-n <nsteps>] [-t <temp>] [-m (0|1)]
                          [-h <hill_height>] [-f <factor>]
```

The gradient file name is provided first, followed by other parameters in any order. They are described below, with their default value in square brackets:

- -n: number of M-C steps to be performed; by default, a minimal number of steps is chosen based on the size of the grid, and the integration runs until a convergence criterion is satisfied (based on the RMSD between the target gradient and the real PMF gradient)

- -t: temperature for M-C sampling (unrelated to the simulation temperature) [500 K]

- -m: use metadynamics-like biased sampling? (0 = false) [1]

- -h: increment for the history-dependent bias ("hill height") [0.01 kcal/mol]

- -f: if non-zero, this factor is used to scale the increment stepwise in the second half of the M-C sampling to refine the free energy estimate [0.5]

Using the default values of all parameters should give reasonable results in most cases.

abf_integrate produces the following output files:

- `<gradient_file>.pmf`: computed free energy surface

- `<gradient_file>.histo`: histogram of M-C sampling (not usable in a straightforward way if the history-dependent bias has been applied)

- `<gradient_file>.est`: estimated gradient of the calculated free energy surface (from finite differences)

- `<gradient_file>.dev`: deviation between the user-provided numerical gradient and the actual gradient of the calculated free energy surface. The RMS norm of this vector field is used as a convergence criteria and displayed periodically during the integration.

Note: Typically, the "deviation" vector field does not vanish as the integration converges. This happens because the numerical estimate of the gradient does not exactly derive from a potential, due to numerical approximations used to obtain it (finite sampling and discretization on a grid).

9.3.2 Metadynamics calculations

Many methods have been introduced in the past that make use of an artificial energy term, that changes and adapts over time, to reconstruct a potential of mean force from a conventional molecular dynamics simulation [24, 22, 45, 16, 31, 25]. One of the most recent, metadynamics, was first designed as a step-wise algorithm, which may be roughly described as an "adaptive umbrella sampling" [31], and was later transformed into a continuous one [27]. The implementation described here provides the latter version only, which is the most commonly used.

In metadynamics, the external potential on the colvars $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_{N_{\mathrm{cv}}})$ is:

$$V_{\mathrm{meta}}(\boldsymbol{\xi}) = \sum_{t'=\delta t, 2\delta t, \ldots}^{t' < t} W \prod_{i=1}^{N_{\mathrm{cv}}} \exp\left(-\frac{(\xi_i - \xi_i(t'))^2}{2\delta_{\xi_i}^2} \right), \tag{25}$$

that is, V_{meta} is a *history–dependent* potential, which acts on the *current* values of the colvars $\boldsymbol{\xi}$ and depends parametrically on the previous values of the colvars. It is constructed as a sum of N_{cv}-dimensional repulsive Gaussian "hills" with a height W: their centers are located at the previously explored configurations $(\boldsymbol{\xi}(\delta t), \boldsymbol{\xi}(2\delta t), \ldots)$, and they extend by approximately $2\delta_{\xi_i}$ in the direction of the i-th colvar.

As the system evolves according to the underlying potential of mean force $A(\boldsymbol{\xi})$ incremented by the metadynamics potential $V_{\mathrm{meta}}(\boldsymbol{\xi})$, new hills will tend to accumulate in the regions with a lower effective free energy $\tilde{A}(\boldsymbol{\xi}) = A(\boldsymbol{\xi}) + V_{\mathrm{meta}}(\boldsymbol{\xi})$. That is, the probability of having a given system configuration $\boldsymbol{\xi}^*$ being explored (and thus, a hill being added there) is proportional to $\exp\left(-\tilde{A}(\boldsymbol{\xi}^*)/\kappa_{\mathrm{B}}T\right)$, which tends to a nearly flat histogram when the simulation is continued until the system has deposited hills across the whole free energy landscape. In this situation, $-V_{\mathrm{meta}}(\boldsymbol{\xi})$ is a good approximant of the free energy $A(\boldsymbol{\xi})$, and the only dependence on the specific conformational history $\boldsymbol{\xi}(\delta t), \boldsymbol{\xi}(2\delta t), \ldots$ is by an irrelevant additive constant:

$$A(\boldsymbol{\xi}) \simeq -V_{\mathrm{meta}}(\boldsymbol{\xi}) + K \tag{26}$$

Provided that the set of collective variables fully describes the relevant degrees of freedom, the accuracy of the reconstructed profile is a function of the ratio between W and δt [10]. For the optimal choice of δ_{ξ_i} and D_{ξ_i}, the diffusion constant of the variable ξ_i, see reference [10]. As a rule of thumb, the very upper limit for the ratio $W/\delta t$ is given by $\kappa_{\mathrm{B}}T/\tau_{\boldsymbol{\xi}}$, where $\tau_{\boldsymbol{\xi}}$ is the longest among

$\boldsymbol{\xi}$'s correlation times. In the most typical conditions, to achieve a good statistical convergence the user would prefer to keep $W/\delta t$ much smaller than $\kappa_{\rm B} T/\tau_{\boldsymbol{\xi}}$.

Given $\Delta\xi$ the extension of the free energy profile along the colvar ξ, and $A^* = A(\boldsymbol{\xi}^*)$ the highest free energy that needs to be sampled (e.g. that of a transition state), the upper bound for the required simulation time is of the order of $N_{\rm s}(\xi) = (A^*\Delta\xi)/(W2\delta_\xi)$ multiples of δt. When several colvars $\boldsymbol{\xi}$ are used, the upper bound amounts to $N_{\rm s}(\xi_1) \times N_{\rm s}(\xi_2) \times \ldots \times N_{\rm s}(\xi_{N_{\rm cv}}) \times \delta t$.

In metadynamics runs performed with this module, the parameter δ_{ξ_i} for each hill (eq. 25) is chosen as half the `width` of the corresponding `colvar` ξ_i, while all the other parameters must be provided within the `metadynamics {...}` block. In addition to the `colvars` option to list the variable to which this bias is applied, the block accepts the following options:

- `name` < (`metadynamics`) Name of this metadynamics instance >
  Acceptable Values: string
  Default Value: "`meta`" + rank number
  Description: This option sets the name for this metadynamics instance. While in general it is not advisable to use more than one metadynamics bias, this allows to distinguish each bias from the others in the output.

- `hillWeight` < (`metadynamics`) Height of each hill (kcal/mol) >
  Acceptable Values: positive decimal
  Default Value: 0.01
  Description: This option sets the height W of the hills that are added during this run.

- `newHillFrequency` < (`metadynamics`) Frequency of hill creation >
  Acceptable Values: positive integer
  Default Value: 100
  Description: This option sets the number of steps after which a new hill is added to the history–dependent potential. It acts on the colvars $\boldsymbol{\xi}$ immediately after it has been added.

- `useGrids` < (`metadynamics`) Optimize the hills calculation >
  Acceptable Values: boolean
  Default Value: on
  Description: This option discretizes all hills on two grids (storing their total energy and gradients, respectively). The grids are defined by `lowerBoundary`, `upperBoundary`, and `width` for each `colvar`, and are saved to the state file. Currently, this is implemented for non–scalar variables only. In this case, `writeHillsTrajectory` may be useful to keep track of the history.

- `gridsUpdateFrequency` < (`metadynamics`) Frequency of update of the grids >
  Acceptable Values: positive integer
  Default Value: `newHillFrequency`
  Description: When `useGrids` is on, all the newly created hills are projected onto the two grids every `gridsUpdateFrequency` steps.

- `dumpFreeEnergyFile` < (`metadynamics`) Periodically save the PMF >
  Acceptable Values: boolean
  Default Value: on
  Description: When `useGrids` and this option are on, the PMF is written every

110

colvarsRestartFrequency steps to the file <outputName>.pmf. If there is more than one metadynamics bias active, the name of this bias is included in the file name.

- saveFreeEnergyFile < (metadynamics) Keep all the PMF files >
  Acceptable Values: boolean
  Default Value: off
  Description: When dumpFreeEnergyFile and this option are on, the step number is included in the file name. Activating this option can be useful to qualitatively assess the convergence of the simulation, by comparing PMFs at different times.

- expandGrids < (metadynamics) Automatically expand the grids >
  Acceptable Values: boolean
  Default Value: off
  Description: When useGrids and this option are on, each time a colvar comes close to the grid boundaries, these are expanded so that there is always a buffer amounting to three times a gaussian width. The new grids have the same spacing as the original ones, and the new boundaries are saved in the state file, which overrides the values from the configuration file.

- hillWidth < (metadynamics) Relative width of the hills >
  Acceptable Values: positive decimal
  Default Value: 3.0
  Description: Along each colvar, the width of each gaussian ($2\delta_{\xi_i}$) is given by the product between this number and the colvar's width. Values larger than 1 act as a "smoothing" parameter. Values smaller than 1 should be avoided, in particular when useGrids is on in order to avoid large errors in the discretization.

- multipleReplicas < (metadynamics) Multiple replicas metadynamics >
  Acceptable Values: boolean
  Default Value: off
  Description: If this option is on, multiple (independent) replica of the same system can be simulated at the same time and share the same hills [37]. This is achieved by letting each replica save its newly created hills to the file "<outputName>.colvars.<name>.<replicaID>.hills": the path to it is communicated to the other replicas through the file replicaFilesRegistry, shared by all replicas. Every replicaUpdateFrequency steps, each replica reads the new hills created by the other replicas and adds them to its own.

- replicaID < (metadynamics) Set the identifier for this replica >
  Acceptable Values: string
  Description: If multipleReplicas is on, this option sets a unique identifier for this replica. Hence, when simulating with more than one replica, different colvars configuration files with different values for this option should be used.

- replicaFilesRegistry < (metadynamics) Multiple replicas database file >
  Acceptable Values: UNIX filename
  Default Value: "<name>.replica_files.txt"
  Description: If multipleReplicas is on, this option sets the path to a replica index file. The names of new hill files containing other replica's new hills are appended to the contents

of this file. Every `replicaUpdateFrequency` steps during a simulation, each replica reads the hills stored in each of the files listed here, except those saved by the replica itself.

- `replicaUpdateFrequency` < (`metadynamics`) Multiple replicas update frequency >
  Acceptable Values: positive integer
  Default Value: `newHillFrequency`
  Description: If `multipleReplicas` is `on`, this option sets the number of steps between updates of the list of hills created by other replicas.

- `writeHillsTrajectory` < (`metadynamics`) Write a log of new hills >
  Acceptable Values: boolean
  Default Value: `on`
  Description: If this option is `on`, a logfile is written by the `metadynamics` bias, with the name "<outputName>.colvars.<name>.hills.traj", which can be useful to follow the time series of the hills. When `multipleReplicas` is `on`, its name changes to "<outputName>.colvars.<name>.<replicaID>.hills.traj".

9.3.3 Harmonic restraints and Steered Molecular Dynamics

The harmonic restraint is allocated by a `harmonic {...}` block, which may contain (in addition to the standard option `colvars`) the following keywords:

- `forceConstant` < (`harmonic`) Harmonic force constant (kcal/mol) >
  Acceptable Values: positive decimal
  Default Value: `1.0`
  Description: This defines the constant for the harmonic potential. In order to cope with multidimensional restraints, it is scaled by the square of the specific `width` for each `colvar` involved. For instance, a force constant of 10 kcal/mol acting on a `dihedral` colvar with a `width` of 2 degrees will amount effectively to 2.5 kcal/mol$\times$degree^{-2}.

- `centers` < (`harmonic`) Harmonic restraints centers >
  Acceptable Values: space–separated list of colvar values
  Description: The centers of the restraint are entered here, and must be of the same number of the requested `colvars`. Each value is in the format of a decimal number if the corresponding colvar returns a scalar, a "(x, y, z)" triplet if it returns a unit vector or a vector, and a "(q0, q1, q2, q3)" quadruplet if it return a rotational quaternion. If a colvar has periodicities or symmetries, its closest image to the restraint center is assumed when calculating the harmonic potential.

- `targets` < (`harmonic`) Harmonic restraints centers >
  Acceptable Values: space–separated list of colvar values
  Description: When defined, the current `centers` will be moved towards these values in `targetsNumSteps` steps, by a linear interpolation. The intermediate values will be evolved by taking into account periodicities or symmetries (the closest image of the target is assumed for each colvar) and intrinsic constraints to the colvar value. For example, when moving the restraint center of a `dihedral`, the total movement in a single run will be at most 180° (a rotation of 210° will be interpreted as one of -150°); if a complete 360° turn of a dihedral is desired, multiple runs (at least two) with different targets should be used. When continuing

112

a simulation run, the `centers` specified in the configuration file <colvarsConfig> will be overridden by those saved in the state file <colvarsInput>.

- `targetsNumSteps` < (`harmonic`) Number of steps for steering >
  Acceptable Values: positive integer
  Default Value: 0
  Description: If larger than zero, it defines the number of steps required to move the restraint centers towards the specified `targets`. After the targets have been reached, the centers are maintained fixed.

To perform Steered Molecular Dynamics (SMD) simulations on the `colvars` implemented in this module, it is sufficient to set a desired set of colvar target in the `harmonic` {...} block, and enable `outputAppliedForce` within the configuration block of each of the `colvars` involved.

9.3.4 Multidimensional histograms

The `histogram` feature is used to record the distribution of a set of collective variables in the form of a N-dimensional histogram. It functions as a "collective variable bias", and is invoked by adding a `histogram` block to the *colvars* configuration file.

In addition to common parameters such as `colvars`, a `histogram` block may define the following parameter:

- `outputFreq` < (`histogram`) Frequency (in timesteps) at which the histogram file is refreshed >
  Acceptable Values: positive integer
  Default Value: Colvar module restart frequency
  Description: The file containing histogram data is written on disk at the given time interval.

Like the ABF bias, the `histogram` feature uses parameters from collective variables to define its grid. The grid ranges from `lowerBoundary` to `upperBoundary`, and the bin width is set by the `width` parameter.

10 Alchemical Free Energy Methods[1]

Alchemical free energy calculations model the physically impossible but computationally realizable process of gradually mutating a subset of atoms of a system from one state to another, through a series of intermediate steps. Two alternative methods for alchemical calculation of free energies from molecular dynamics simulation are available in NAMD: Free energy perturbation (FEP) and thermodynamic integration (TI).

10.1 Theoretical Background

Free energy differences can be obtained through four different routes: (i) probability densities, (ii) free energy perturbation, (iii) thermodynamic integration, or (iv) nonequilibrium work approaches [13]. Within NAMD, alchemical transformations are modeled following the second and the third routes, both of which rely upon the use of a general extent parameter often referred to as the coupling parameter [6, 34, 28, 29] for the description of chemical changes in the molecular systems between the reference and the target states.

10.1.1 The dual–topology paradigm

In a typical alchemical transformation setup involving the alteration of one chemical species into an alternate one in the course of the simulation, the atoms in the molecular topology can be classified into three groups, (i) a group of atoms that do not change during the simulation — *e.g.* the environment, (ii) the atoms describing the reference state, a, of the system, and (iii) the atoms that correspond to the target state, b, at the end of the alchemical transformation. The atoms representative of state a should *never* interact with those of state b throughout the MD simulation. Such a setup, in which atoms of both the initial and the final states of the system are present in the molecular topology file — *i.e.* the `psf` file — is characteristic of the so–called "dual topology" paradigm [20, 36, 2]. The hybrid Hamiltonian of the system is a function of the general extent parameter, λ, which connects smoothly state a to state b. In the simplest case, such a connection may be achieved by linear combination of the corresponding Hamiltonians:

$$\mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda) = \mathcal{H}_0(\mathbf{x}, \mathbf{p}_x) + \lambda \mathcal{H}_b(\mathbf{x}, \mathbf{p}_x) + (1 - \lambda)\mathcal{H}_a(\mathbf{x}, \mathbf{p}_x) \tag{27}$$

where $\mathcal{H}_a(\mathbf{x}, \mathbf{p}_x)$ describes the interaction of the group of atoms representative of the reference state, a, with the rest of the system. $\mathcal{H}_b(\mathbf{x}, \mathbf{p}_x)$ characterizes the interaction of the target topology, b, with the rest of the system. $\mathcal{H}_0(\mathbf{x}, \mathbf{p}_x)$ is the Hamiltonian describing those atoms that do not undergo any transformation during the MD simulation.

For instance, in the point mutation of an alanine side chain into that of glycine, by means of a free energy calculation — either free energy perturbation or thermodynamic integration, the topology of both the methyl group of alanine and the hydrogen borne by the C_α in glycine co–exist throughout the simulation (see Figure 7), yet without actually seeing each other.

The energy and forces are defined as a function of λ, in such a fashion that the interaction of the methyl group of alanine with the rest of the protein is effective at the beginning of the simulation, *i.e.* $\lambda = 0$, while the glycine C_α hydrogen atom does not interact with the rest of the protein, and *vice versa* at the end of the simulation, *i.e.* $\lambda = 1$. For intermediate values of λ, both the alanine

---
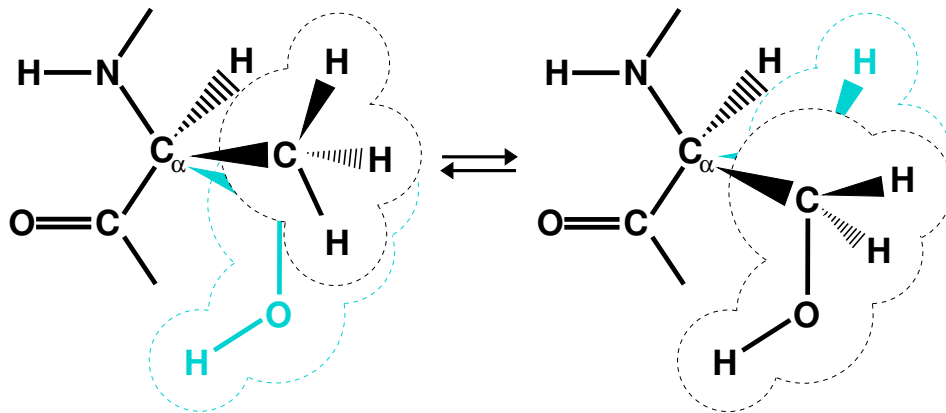
Figure 7: Dual topology description for an alchemical simulation. Case example of the mutation of alanine into serine. The lighter color denotes the non–interacting, alternate state.

and the glycine side chains participate in nonbonded interactions with the rest of the protein, scaled on the basis of the current value of λ. It should be clearly understood that these side chains never interact with each other.

Noteworthily, end points of alchemical transformations carried out in the framework of the dual–topology paradigm have been shown to be conducive to numerical instabilities from molecular dynamics simulations, often coined as "end–point catastrophes". These scenarios are prone to occur when λ becomes close to 0 or 1, and incoming atoms instantly appear where other particles are already present, which results in a virtually infinite potential as the interatomic distance tends towards 0. Such "end–point catastrophes" can be profitably circumvented by introducing a so–called soft–core potential [5], aimed at a gradual scaling of the short–range nonbonded interactions of incoming atoms with their environment. What is really being modified is the value of the coupling parameter that scales the interactions — *i.e.*, if set to 0, the latter are off; if set to 1, they are on — in lieu of the actual value of λ provided by the user.

It is also worth noting that the free energy calculation does not alter intermolecular bonded potentials, *e.g.* bond stretch, valence angle deformation and torsions, in the course of the simulation. In calculations targeted at the estimation of free energy differences between two states characterized by distinct environments — *e.g.* a ligand, bound to a protein in the first simulation, and solvated in water, in the second — as is the case for most free energy calculations that make use of a thermodynamic cycle, perturbation of intramolecular terms may, by and large, be safely avoided [7].

10.1.2 Free Energy Perturbation

Within the FEP framework [6, 12, 13, 21, 30, 34, 43, 44, 46], the free energy difference between two alternate states, a and b, is expressed by:

$$\Delta A_{a \to b} = -\frac{1}{\beta} \ln \left\langle \exp\left\{ -\beta \left[\mathcal{H}_b(\mathbf{x}, \mathbf{p}_x) - \mathcal{H}_a(\mathbf{x}, \mathbf{p}_x) \right] \right\} \right\rangle_a \tag{28}$$

Here, $\beta^{-1} \equiv k_B T$, where k_B is the Boltzmann constant, T is the temperature. $\mathcal{H}_a(\mathbf{x}, \mathbf{p}_x)$ and $\mathcal{H}_b(\mathbf{x}, \mathbf{p}_x)$ are the Hamiltonians describing states a and b, respectively. $\langle \cdots \rangle_a$ denotes an ensemble average over configurations representative of the initial, reference state, a.

Convergence of equation (28) implies that low–energy configurations of the target state, b,
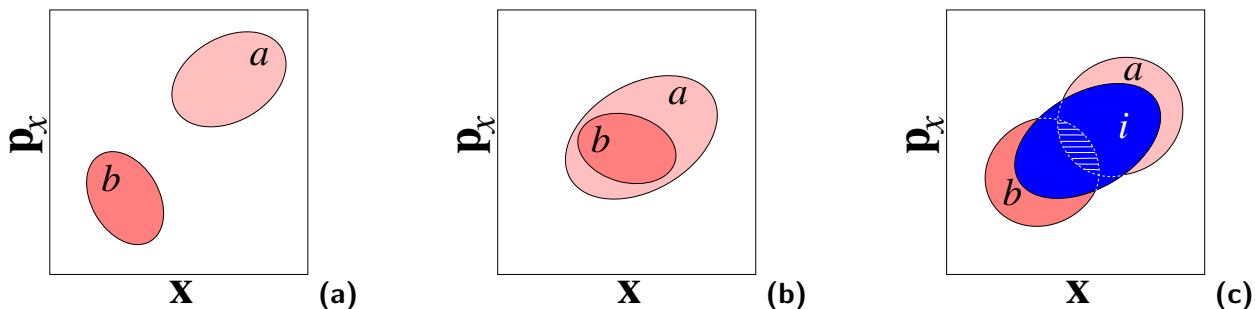
Figure 8: Convergence of an FEP calculation. If the ensembles representative of states a and b are too disparate, equation (28) will not converge **(a)**. If, in sharp contrast, the configurations of state b form a subset of the ensemble of configurations characteristic of state a, the simulation is expected to converge **(b)**. The difficulties reflected in case **(a)** may be alleviated by the introduction of mutually overlapping intermediate states that connect a to b **(c)**. It should be mentioned that in practice, the kinetic contribution, $\mathcal{T}(\mathbf{p}_x)$, is assumed to be identical for state a and state b.

are also configurations of the reference state, a, thus resulting in an appropriate overlap of the corresponding ensembles — see Figure 8. Transformation between the two thermodynamic states is replaced by a series of transformations between non–physical, intermediate states along a well–delineated pathway that connects a to b. This pathway is characterized by the general extent parameter [6, 28, 29, 34], λ, that makes the Hamiltonian and, hence, the free energy, a continuous function of this parameter between a and b:

$$\Delta A_{a \rightarrow b} = -\frac{1}{\beta} \sum_{i=1}^{N} \ln \left\langle \exp \left\{ -\beta \left[\mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda_{i+1}) - \mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda_i) \right] \right\} \right\rangle_i \tag{29}$$

Here, N stands for the number of intermediate stages, or "windows" between the initial and the final states — see Figure 8.

10.1.3 Thermodynamic Integration

An alternative to the perturbation formula for free energy calculation is Thermodynamic Integration (TI). With the TI method, the free energy is given as [29, 42, 19]:

$$\Delta A = \int_0^1 \left\langle \frac{\partial \mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda)}{\partial \lambda} \right\rangle_\lambda d\lambda \tag{30}$$

In the multi-configuration thermodynamic integration approach [42] implemented in NAMD, $\langle \partial \mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda)/\partial \lambda \rangle_\lambda$, the ensemble average of the derivative of the internal energy with respect to λ, is collected for a series of discrete λ values and written to `tiOutFile`. These values are analyzed by the separately distributed script `NAMD_ti.pl`, which performs the integration of individual energy components and reports back the total ΔA values for the transformation.

10.2 Implementation of the free energy methods in NAMD

The procedures implemented in NAMD are particularly adapted for performing free energy cal-culations that split the λ reaction path into a number of non–physical, intermediate states, or

"windows". Separate simulations can be started for each window. Alternatively, the TCL scripting ability of NAMD can be employed advantageously to perform the complete simulation in a single run. An example, making use of such a script, is supplied at the end of this section.

The following keywords can be used to run alchemical free energy calculations, whether FEP or TI. Each keyword has two variants: one that is used in FEP calculations and a second that should be used when performing TI. The FEP and TI variants are seperated by slashes like so: `fepKeyword / tiKeyword`, listing first the FEP keyword and second the TI keyword. The FEP and TI specific keywords should be used exclusively with FEP or TI, respectively, and should not be intermixed during a simulation.

- `fepOn / thermInt` < Is an alchemical free energy perturbation to be performed? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Turns on alchemical transformation methods in NAMD.

- `lambda / tilambda` < Current value of the coupling parameter >
  Acceptable Values: positive decimal between 0.0 and 1.0
  Description: The coupling parameter value determining the progress of the perturbation for FEP or TI.

- `lambda2 / Not applicable` < Forward projected value of the coupling parameter >
  Acceptable Values: positive decimal between 0.0 and 1.0
  Description: The `lambda2` value corresponds to the coupling parameter to be used for sampling in the next window. The free energy difference between `lambda2` and `lambda` is calculated. Through simulations at progressive values of `lambda` and `lambda2` the total free energy difference may be determined.

- `fepEquilSteps / tiEquilSteps` < Number of equilibration steps in a window, prior to data collection >
  Acceptable Values: positive integer less than `numSteps` or `run`
  Default Value: 0
  Description: In each window `fepEquilSteps/tiEquilSteps` steps of equilibration can be performed before ensemble averaging is initiated. The output also contains the data gathered during equilibration and is meant for analysis of convergence properties of the alchemical free energy calculation.

- `fepFile / tiFile` < pdb file with perturbation flags >
  Acceptable Values: filename
  Default Value: coordinates
  Description: `pdb` file to be used for indicating the status of all atoms pertaining to the system, with respect to the alchemical transformation. If this parameter is not declared specifically, then the `pdb` file specified by `coordinates` is utilized for this information.

- `fepCol / tiCol` < Column in the `fepFile/tiFile` that carries the perturbation flag >
  Acceptable Values: X, Y, Z, O or B
  Default Value: B
  Description: Column of the `pdb` file to use for retrieving the status of each atom, *i.e.* a flag that indicates which atom will be perturbed in the course of the alchemical transformation.

117

A value of `-1` in the specified column indicates that the atom will vanish as λ moves from 0 to 1, whereas a value of `1` indicates that it will grow.

- `fepOutFreq / tiOutFreq` < Frequency of free energy output in time–steps >
  Acceptable Values: positive integer
  Default Value: 5
  Description: Every `fepOutFreq/tiOutFreq` number of MD steps, the output file `fepOutFile/tiOutFile` is updated by dumping energies that are used for ensemble averaging. This variable could be set to `1` to include all the configurations for ensemble averaging. Yet, it is recommended to update `fepOutFile/tiOutFile` energies at longer intervals to avoid large files containing highly correlated data, unless a post–treatment, *e.g.* Bennett's acceptance ratio (BAR) [3] or simple overlap sampling (SOS) [32], is to be performed.

- `fepOutFile / tiOutFile` < Alchemical free energy output filename >
  Acceptable Values: filename
  Default Value: outfilename
  Description: An output file named `fepOutFile`, containing the FEP energies, or `tiOutFile`, containing the TI derivatives, dumped every `fepOutFreq/tiOutFreq` steps.

- `fepVdwShiftCoeff / tiVdwShiftCoeff` < Soft-core van der Waals radius-shifting coefficient >
  Acceptable Values: positive decimal
  Default Value: 0
  Description: This is a radius-shifting coefficient of λ that is used to construct the modified vdW interactions during alchemical free energy calculations. Providing a positive value for `fepVdWShiftCoeff/tiVdwShiftCoeff` ensures that the vdW potential is finite everywhere for small values of λ, which significantly improves the accuracy and convergence of FEP and TI calculations, and also prevents overlapping particles from making the simulation unstable. During FEP and TI, assuming $\lambda = 0$ denotes an absence of interaction, the interatomic distances used in the Lennard-Jones potential are shifted according to [5, 33]:
  $$r^2 \rightarrow r^2 + \texttt{fepVdWShiftCoeff} \times (1 - \lambda)$$

- `fepElecLambdaStart / tiElecLambdaStart` < Value of λ to introduce electrostatic interactions >
  Acceptable Values: positive decimal
  Default Value: 0
  Description: In order to avoid the so–called "end-point catastrophes", it is crucial to avoid situations where growing particles overlap with existing particles with an unbounded interaction potential, which would approach infinity as the interaction distance approaches zero [5, 13]. One possible route for avoiding overlap of unbounded electrostatic potentials consists of allowing a bounded (soft-core) vdW potential, using a positive value of `fepVdWShiftCoeff/tiVdwShiftCoeff`, to repel first all overlapping particles at low values of λ. As λ increases, once the particles are repelled, it becomes safe to turn on FEP or TI electrostatics. `fepElecLambdaStart/tiElecLambdaStart` is the value of the λ at which electrostatic interactions are turned on and linearly increased towards their nominal value. Below `fepElecLambdaStart/tiElecLambdaStart`, electrostatic interactions are turned off for all incoming and outgoing particles.

- `fepVdwLambdaEnd / tiVdwLambdaEnd` < Value of λ to cancel van der Waals interactions >

  Acceptable Values: positive decimal
  Default Value: 1.0
  Description: If the above `fepElecLambdaStart/tiElecLambdaStart` option is being used, it may be further desirable to separate the scaling of van der Waals and electrostatic interactions. `fepVdwLambdaEnd/tiVdwLambdaEnd` sets the value of λ above which all vdW interactions are fully enabled.

- `decouple / decouple` < Disable scaling of nonbonded interactions within alchemical partitions >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: With `decouple` set to `on`, only nonbonded interactions of perturbed, incoming and outgoing atoms with their environment are scaled, while interactions within the subset of perturbed atoms are preserved. On the contrary, if `decouple` is set to `off`, interactions within the perturbed subset of atoms are also scaled and contribute to the cumulative free energy. In most calculations, intramolecular annihilation free energies are not particularly informative, and decoupling ought to be preferred. Under certain circumstances, it may, however, be desirable to scale intramolecular interactions, provided that the latter are appropriately accounted for in the thermodynamic cycle [13].

10.3   Examples of input files for running alchemical free energy calculations

The first example illustrates the use of TCL scripting for running an alchemical transformation with the FEP feature of NAMD. In this calculation, λ is changed continuously from 0 to 1 by increments of $\delta\lambda = 0.1$.

`fepOn` `On`	Turn FEP functionality on.
`fepFile` `ion.fep`	File containing the information about growing/shrinking atoms described in column `X`.
`fepCol` `X`	
`fepOutfile` `ion.fepout`	Output file containing the free energy.
`fepOutFreq` `5`	Frequency at which `fepOutFreq` is updated.
`fepEquilSteps` `5000`	Number of equilibration steps per λ–state.
`set Lambda0` `0.0`	Starting value of λ.
`set dLambda 0.1`	Increment of λ, *i.e.* $\delta\lambda$.

```
while {$Lambda0 <= 1.0} {
 lambda $Lambda0
 set Lambda0 [expr \$Lambda0 + \$dLambda]
 lambda2 $Lambda0
 run  10000
}
```

TCL script to increment λ:
(1) set `lambda` value;
(2) increment λ;
(3) set `lambda2` value;
(4) run 10,000 MD steps.

The user should be reminded that by setting `run 10000`, 10,000 MD steps will be performed, which includes the preliminary `fepEquilSteps` equilibration steps. This means that here, the ensemble average of equation (29) will be computed over 5,000 MD steps.

Alternatively, λ–states may be declared explicitly, avoiding the use of TCL scripting:

```
lambda          0.0                (1) set lambda value;
lambda2         0.1                (2) set lambda2 value;
run             10000              (3) run 10,000 MD steps.
```

This option is generally preferred to set up windows of diminishing widths as $\lambda \to 0$ or 1 — a way to circumvent end–point singularities caused by appearing atoms that may clash with their surroundings.

The following second input is proposed for the measuring via TI the free energy of a particle insertion.

```
thermInt       On                 # Enable thermodynamic integration
tiFile         ion.alch.pdb       # PDB file with perturbation flags
tiCol          B                  # Perturbation flags in Beta column
tiOutfile      ion.ti.out
tiOutFreq      5
tiEquilSteps   5000


tiVdWShiftCoeff    1              # Enable soft-core vdW potential
tiElecLambdaStart  0.1           # Introduce electrostatics for lambda > 0.1
tiLambda 0
run 10000
tiLambda 0.00001
run 10000
tiLambda 0.0001
run 10000
tiLambda 0.001
run 10000
tiLambda 0.01
run 10000


set Lambda          0.1

while {$Lambda <= 0.9} {
  tiLambda $Lambda
  run 10000
  set Lambda [expr $Lambda + 0.1]
}

tiLambda 0.99
run 10000
tiLambda 0.999
run 10000
tiLambda 0.9999
run 10000
tiLambda 0.99999
run 10000
tiLambda 1
run 10000
```

Robust sampling of the free energy of particle insertion is enabled by the use of soft-core van der Waals scaling with the `VdWShiftCoeff` parameter, delayed introduction of electrostatics with a non-zero `tiElecLambdaStart` value, and very gradual scaling of λ towards its end points.

10.4 Description of a free energy calculation output

10.4.1 Free Energy Perturbation

When running FEP, the `fepOutFile` contains electrostatic and van der Waals energy data calculated for `lambda` and `lambda2`, written every `fepOutFreq` steps. The column `dE` is the energy difference of the single configuration, `dE_avg` and `dG` are the instantaneous ensemble average of the energy and the calculated free energy at the time step specified in column 2, respectively. The temperature is specified in the penultimate column. Upon completion of `fepEquilSteps` steps, the calculation of `dE_avg` and `dG` is restarted. The accumulated net free energy change is written at each lambda value and at the end of the simulation.

Whereas the FEP module of NAMD supplies free energy differences determined from equation (28), the wealth of information available in `fepOutFile` may be utilized profitably to explore different routes towards the estimation of ΔA. Both BAR and SOS methods, which combine advantageously *direct* and *reverse* transformations to improve convergence and accuracy of the calculation, represent relevant alternatives to brute–force application of the FEP formula [32].

Within the SOS framework, the free energy difference between states λ_i and λ_{i+1} is expressed as:

$$\exp(-\beta\Delta A_{i\rightarrow i+1}) = \frac{\left\langle \exp\left\{-\dfrac{\beta}{2}\left[\mathcal{H}(\mathbf{x},\mathbf{p}_x;\lambda_{i+1}) - \mathcal{H}(\mathbf{x},\mathbf{p}_x;\lambda_i)\right]\right\}\right\rangle_i}{\left\langle \exp\left\{-\dfrac{\beta}{2}\left[\mathcal{H}(\mathbf{x},\mathbf{p}_x;\lambda_i) - \mathcal{H}(\mathbf{x},\mathbf{p}_x;\lambda_{i+1})\right]\right\}\right\rangle_{i+1}} \tag{31}$$

and can be readily used with the statistical information provided by the forward and the backward runs.

10.4.2 Thermodynamic Integration

When running TI free energy calculations, the `elec_dU/dl` and `vdW_dU/dl` values reported in `tiOutFile` are the derivatives of the internal energy with respect to λ — *i.e.* $\frac{\partial U}{\partial \lambda}$ for electrostatics and, van der Waals, respectively. `dU/dl` values are averages over the last `tiOutFreq` steps. Cumulative averages for each component are reported alongside in the `_avg` columns.

The electrostatics and vdW are separated following a partition scheme — that is, the "appearing" and the "disappearing" atoms are accounted for separately. "Partition 1" contains those atoms whose interactions are switched up as λ increases — *i.e.* flagged with `1` in the `fepFile` or `tiFile`. "Partition 2" represents those atoms whose interactions are switched down as λ increases — *i.e.* flagged with `-1`. ΔA values for each component are obtained by integrating from $\lambda = 0$ to 1 using the respective `ELEC / VDW LAMBDA` listed for each partition after the title.

Analysis is handled by the `NAMD_ti` script, available from

http://www.ks.uiuc.edu/Research/namd/utilities/

Although the output format of `NAMD_ti.pl` may appear to lend itself easily to interpretation of the individual contributions to the free energy total (elec and vdW for each partition), this is rarely appropriate as these values are path-dependent. For example, an output such as

```
|------------------------------------------------|
|          |   elec   |    vdW   |   Subtotal  |
|------------------------------------------------|
| Part. 1 |  -0.5748 |  -6.3452 |    -6.9200  |
| Part. 2 |   0.5391 |   4.9387 |     5.4778  |
|------------------------------------------------|
| Subtotal|   0.6048 |   0.3293 |   -12.3978  |
|------------------------------------------------|
Total deltaG for transition lambda 0 -> 1: -12.3978
```

may encourage interpretations along the lines of "the free energy for switching on the van der Waals interactions for the atoms of partition 1 was -6.35kcal/mol". This is only correct in the very narrow context of the simulation setup and parameters used in this case and is not informative in a broader sense.

The choice of λ values will depend on the application, but in general it is important to examine the shape of the curve to ensure that sampling is adequate to give a good estimate of the integral. In particular, it will be necessary to sample more finely towards the end points in order to accurately account for the strong repulsive van der Waals forces encountered when inserting particles into a system (see Figure 9).
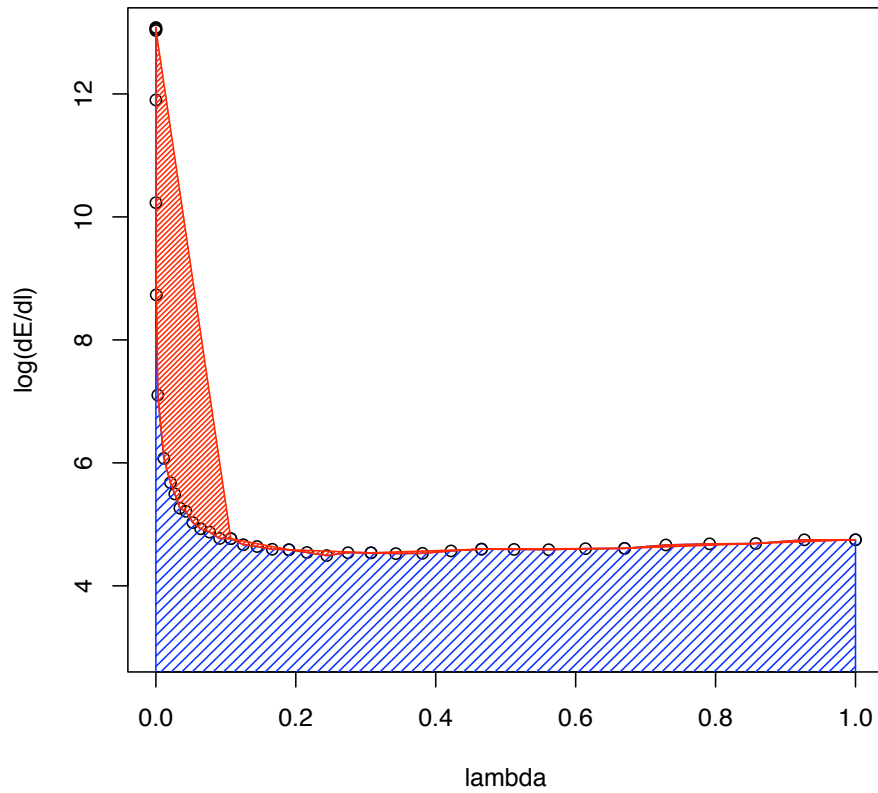
Figure 9: Sample TI data ($log(\langle \frac{\partial U}{\partial \lambda} \rangle)$) against λ). The blue shaded area shows the integral with fine sampling close to the end point. The red area shows the difference when λ values are more sparse. In this example, insufficient sampling before $\lambda \simeq 0.1$ can result in a large overestimation of the integral. Beyond $\simeq 0.2$, sparser sampling is justified as dE/dλ is not changing quickly.

11 Accelerated Sampling Methods

11.1 Locally enhanced sampling

Locally enhanced sampling (LES) [38, 40, 41] increases sampling and transition rates for a portion of a molecule by the use of multiple non-interacting copies of the enhanced atoms. These enhanced atoms experience an interaction (electrostatics, van der Waals, and covalent) potential that is divided by the number of copies present. In this way the enhanced atoms can occupy the same space, while the multiple instances and reduces barriers increase transition rates.

11.1.1 Structure generation

To use LES, the structure and coordinate input files must be modified to contain multiple copies of the enhanced atoms. `psfgen` provides the `multiply` command for this purpose. NAMD supports a maximum of 15 copies, which should be sufficient.

Begin by generating the complete molecular structure and guessing coordinates as described in Sec. 4. As the last operation in your script, prior to writing the psf and pdb files, add the `multiply` command, specifying the number of copies desired and listing segments, residues, or atoms to be multiplied. For example, `multiply 4 BPTI:56 BPTI:57` will create four copies of the last two residues of segment BPTI. You must include all atoms to be enhanced in a single `multiply` command in order for the bonded terms in the psf file to be duplicated correctly. Calling `multiply` on connected sets of atoms multiple times will produce unpredictable results, as may running other commands after `multiply`.

The enhanced atoms are duplicated exactly in the structure—they have the same segment, residue, and atom names. They are distinguished only by the value of the B (beta) column in the pdb file, which is 0 for normal atoms and varies from 1 to the number of copies created for enhanced atoms. The enhanced atoms may be easily observed in VMD with the atom selection `beta != 0`.

11.1.2 Simulation

In practice, LES is a simple method used to increase sampling; no special output is generated. The following parameters are used to enable LES:

- `les` < is locally enhanced sampling active? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether or not LES is active.

  `lesFactor` < number of LES images to use >
  Acceptable Values: positive integer equal to the number of images present
  Description: This should be equal to the factor used in `multiply` when creating the structure. The interaction potentials for images is divided by `lesFactor`.

- `lesReduceTemp` < reduce enhanced atom temperature? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Enhanced atoms experience interaction potentials divided by `lesFactor`. This allows them to enter regions that would not normally be thermally accessible. If this is not desired, then the temperature of these atoms may be reduced to correspond with the

reduced potential. This option affects velocity initialization, reinititialization, reassignment, and the target temperature for langevin dynamics. Langevin dynamics is recommended with this option, since in a constant energy simulation energy will flow into the enhanced degrees of freedom until they reach thermal equilibrium with the rest of the system. The reduced temperature atoms will have reduced velocities as well, unless `lesReduceMass` is also enabled.

- `lesReduceMass` < reduce enhanced atom mass? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Used with `lesReduceTemp` to restore velocity distribution to enhanced atoms. If used alone, enhanced atoms would move faster than normal atoms, and hence a smaller timestep would be required.

-

- `lesFile` < PDB file containing LES flags >
  Acceptable Values: UNIX filename
  Default Value: `coordinates`
  Description: PDB file to specify the LES image number of each atom. If this parameter is not specified, then the PDB file containing initial coordinates specified by `coordinates` is used.

- `lesCol` < column of PDB file containing LES flags >
  Acceptable Values: `X`, `Y`, `Z`, `O`, or `B`
  Default Value: `B`
  Description: Column of the PDB file to specify the LES image number of each atom. This parameter may specify any of the floating point fields of the PDB file, either X, Y, Z, occupancy, or beta-coupling (temperature-coupling). A value of 0 in this column indicates that the atom is not enhanced. Any other value should be a positive integer less than `lesFactor`.

11.2 Replica exchange simulations

The `lib/replica/` directory contains Tcl scripts that implement replica exchange for NAMD, using a Tcl server and socket connections to drive a separate NAMD process for every replica used in the simulation. Replica exchanges and energies are recorded in the potenergy.dat, realtemp.dat, and targtemp.dat files written in the output directory. These can be viewed with, e.g., "`xmgrace -nxy ....potenergy.dat`" There is also a script to load the output into VMD and color each frame according to target temperature. An example simulation folds a 66-atom model of a deca-alanine helix in about 10 ns.

This implementation is designed to be modified by the user to implement exchanges of parameters other than temperature or via other temperature exchange methods. The scripts should provide a good starting point for any simulation method requiring a number of loosely interacting systems.

`replica_exchange.tcl` is the master Tcl script for replica exchange simulations, it is run in `tclsh` *outside of NAMD* and takes a replica exchange config file as an argument:

```
tclsh ../replica_exchange.tcl fold_alanin.conf
tclsh ../replica_exchange.tcl restart_1.conf
```

`replica_exchange.tcl` uses code in `namd_replica_server.tcl`, a general script for driving NAMD slaves, and `spawn_namd.tcl`, a variety of methods for launching NAMD slaves.

`show_replicas.vmd` is a script for loading replicas into VMD; first source the replica exchange conf file and then this script, then repeat for each restart conf file or for example just do "`vmd -e load_all.vmd`". This script will likely destroy anything else you are doing in VMD at the time, so it is best to start with a fresh VMD. `clone_reps.vmd` provides the `clone_reps` commmand to copy graphical representation from the top molecule to all other molecules.

A replica exchange config file should define the following Tcl variables:

- `num_replicas`, the number of replica simulations to use,

- `min_temp`, the lowest replica target temperature,

- `max_temp`, the highest replica target temperature,

- `steps_per_run`, the number of steps between exchange attempts,

- `num_runs`, the number of runs before stopping (should be divisible by `runs_per_frame` $\times$ `frames_per_restart`).

- `runs_per_frame`, the number of runs between trajectory outputs,

- `frames_per_restart`, the number of frames between restart outputs,

- `namd_config_file`, the NAMD config file containing all parameters, needed for the simulation except `seed`, `langevin`, `langevinDamping`, `langevinTemp`, `outputEnergies`, `outputname`, `dcdFreq`, `temperature`, `bincoordinates`, `binvelocities`, or `extendedSystem`, which are provided by `replica_exchange.tcl`,

- `output_root`, the directory/fileroot for output files,

- `psf_file`, the psf file for `show_replicas.vmd`,

- `initial_pdb_file`, the initial coordinate pdb file for `show_replicas.vmd`,

- `fit_pdb_file`, the coodinates that frames are fit to by `show_replicas.vmd` (e.g., a folded structure),

- `server_port`, the port to connect to the replica server on, and

- `spawn_namd_command`, a command from `spawn_namd.tcl` and arguments to launch NAMD jobs.

The `lib/replica/example/` directory contains all files needed to fold a 66-atom model of a deca-alanine helix:

- `alanin_base.namd`, basic config options for NAMD,

- `alanin.params`, parameters,

- `alanin.psf`, structure,

- `unfolded.pdb`, initial coordinates,

- `alanin.pdb`, folded structure for fitting in `show_replicas.vmd`,

- `fold_alanin.conf`, config file for `replica_exchange.tcl` script,

- `restart_1.conf`, config file to continue alanin folding another 10 ns, and

- `load_all.vmd`, load all output into VMD and color by target temperature.

The `fold_alanin.conf` config file contains the following settings:

```
set num_replicas 8
set min_temp 300
set max_temp 600
set steps_per_run 1000
set num_runs 10000
set runs_per_frame 10
set frames_per_restart 10
set namd_config_file "alanin_base.namd"
set output_root "output/fold_alanin" ; # directory must exist
set psf_file "alanin.psf"
set initial_pdb_file "unfolded.pdb"
set fit_pdb_file "alanin.pdb"
set namd_bin_dir /Projects/namd2/bin/current/Linux64
set server_port 3177
set spawn_namd_command \
  [list spawn_namd_ssh "cd [pwd]; [file join $namd_bin_dir namd2] +netpoll" \
  [list beirut belfast] ]
```

12 Runtime Analysis

12.1 Pair interaction calculations

NAMD supportes the calculation of interaction energy calculations between two groups of atoms. When enabled, pair interaction information will be calculated and printed in the standard output file on its own line at the same frequency as energy output. The format of the line is `PAIR INTERACTION: STEP:` *step* `VDW_FORCE:` *fx fy fz* `ELECT_FORCE:` *fx fy fz*. The displayed force is the force on atoms in group 1 and is units of kcal/mol/Å.

For trajectory analysis the recommended way to use this set of options is to use the NAMD Tcl scripting interface as described in Sec. 2.2.2 to run for 0 steps, so that NAMD prints the energy without performing any dynamics.

- **pairInteraction** < is pair interaction calculation active? >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: Specifies whether pair interaction calculation is active.

- **pairInteractionFile** < PDB file containing pair interaction flags >
  Acceptable Values: UNIX filename
  Default Value: `coordinates`
  Description: PDB file to specify atoms to use for pair interaction calculations. If this parameter is not specified, then the PDB file containing initial coordinates specified by `coordinates` is used.

- **pairInteractionCol** < column of PDB file containing pair interaction flags >
  Acceptable Values: `X`, `Y`, `Z`, `O`, or `B`
  Default Value: `B`
  Description: Column of the PDB file to specify which atoms to use for pair interaction calculations. This parameter may specify any of the floating point fields of the PDB file, either X, Y, Z, occupancy, or beta-coupling (temperature-coupling).

- **pairInteractionSelf** < compute within-group interactions instead of bewteen groups >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: When active, NAMD will compute bonded and nonbonded interactions only for atoms within group 1.

- **pairInteractionGroup1** < Flag to indicate atoms in group 1? >
  Acceptable Values: integer
  Description:

- **pairInteractionGroup2** < Flag to indicate atoms in group 2? >
  Acceptable Values: integer
  Description: These options are used to indicate which atoms belong to each interaction group. Atoms with a value in the column specified by `pairInteractionCol` equal to `pairInteractionGroup1` will be assigned to group 1; likewise for group 2.

12.2 Pressure profile calculations

NAMD supports the calculation of lateral pressure profiles as a function of the z-coordinate in the system. The algorithm is based on that of Lindahl and Edholm (JCP 2000), with modifications to enable Ewald sums based on Sonne et al (JCP 122, 2005).

The simulation space is partitioned into slabs, and half the virial due to the interaction between two particles is assigned to each of the slabs containing the particles. This amounts to employing the Harasima contour, rather than the Irving-Kirkwood contour, as was done in NAMD 2.5. The diagonal components of the pressure tensor for each slab, averaged over all timesteps since the previous output, are recorded in the NAMD output file. The units of pressure are the same as in the regular NAMD pressure output; i.e., bar.

The total virial contains contributions from up to four components: kinetic energy, bonded interactions, nonbonded interactions, and an Ewald sum. All but the Ewald sums are computed online during a normal simulation run (this is a change from NAMD 2.5, when nonbonded contributions to the Ewald sum were always computed offline). If the simulations are performed using PME, the Ewald contribution should be estimated using a separate, offline calculation based on the saved trajectory files. The nonbonded contribution using a cutoff different from the one used in the simulation may also be computed offline in the same fashion as for Ewald, if desired.

Pressure profile calculations may be performed in either constant volume or constant pressure conditions. If constant pressure is enabled, the slabs thickness will be rescaled along with the unit cell; the dcdUnitCell option will also be switched on so that unit cell information is stored in the trajectory file.

NAMD 2.6 now reports the lateral pressure partitioned by interaction type. Three groups are reported: kinetic + rigid bond restraints (referred to as "internal", bonded, and nonbonded. If Ewald pressure profile calculations are active, the Ewald contribution is reported in the nonbonded section, and no other contributions are reported.

NAMD 2.6 also permits the pressure profile to be partitioned by atom type. Up to 15 atom groups may be assigned, and individual contribution of each group (for the "internal" pressures) and the pairwise contributions of interactions within and between groups (for the nonbonded and bonded pressures) are reported in the output file.

- **pressureProfile** $<$ compute pressure profile $>$
  Acceptable Values: on or off
  Default Value: off
  Description: When active, NAMD will compute kinetic, bonded and nonbonded (but not reciprocal space) contributions to the pressure profile. Results will be recorded in the NAMD output file in lines with the format PRESSUREPROFILE: ts Axx Ayy Azz Bxx Byy Bzz ... , where ts is the timestep, followed by the three diagonal components of the pressure tensor in the first slab (the slab with lowest z), then the next lowest slab, and so forth. The output will reflect the pressure profile averaged over all the steps since the last output.

  NAMD also reports kinetic, bonded and nonbonded contributions separately, using the same format as the total pressure, but on lines beginning with PPROFILEINTERNAL, PPROFILEBONDED, and PPROFILENONBONDED.

- **pressureProfileSlabs** $<$ Number of slabs in the spatial partition $>$
  Acceptable Values: Positive integer
  Default Value: 10

Description: NAMD divides the entire periodic cell into horizontal slabs of equal thickness; `pressureProfileSlabs` specifies the number of such slabs.

- `pressureProfileFreq` < How often to output pressure profile data >
  Acceptable Values: Positive integer
  Default Value: 1
  Description: Specifies the number of timesteps between output of pressure profile data.

- `pressureProfileEwald` < Enable pressure profile Ewald sums >
  Acceptable Values: `on` or `off`
  Default Value: `off`
  Description: When enabled, only the Ewald contribution to the pressure profile will be computed. For trajectory analysis the recommended way to use this option is to use the NAMD Tcl scripting interface as described in Sec. 2.2.2 to run for 0 steps, so that NAMD prints the pressure profile without performing any dynamics.

  The Ewald sum method is as described in Sonne et al. (JCP 122, 2005). The number of k vectors to use along each periodic cell dimension is specified by the `pressureProfileEwald`n parameters described below.

- `pressureProfileEwaldX` < Ewald grid size along X >
  Acceptable Values: Positive integer
  Default Value: 10
  Description:

- `pressureProfileEwaldY` < Ewald grid size along Y >
  Acceptable Values: Positive integer
  Default Value: 10
  Description:

- `pressureProfileEwaldZ` < Ewald grid size along Z >
  Acceptable Values: Positive integer
  Default Value: 10
  Description:

- `pressureProfileAtomTypes` < Number of atom type partitions >
  Acceptable Values: Positive integer
  Default Value: 1
  Description: If `pressureProfileAtomTypes` is greater than 1, NAMD will calculate the separate contributions of each type of atom to the internal, bonded, nonbonded, and total pressure. In the case of the internal contribution, there will be n pressure profile data sets reported on each `PPROFILEINTERNAL` line, where n is the number of atom types. All the partial pressures for atom type 1 will be followed by those for atom type 2, and so forth. The other three pressure profile reports will contain $n(n + 1)/2$ data sets. For example, if there are $n = 3$ atom types, the six data sets arising from the three inter-partition and the three intra-partition interactions will be reported in the following order: 1–1, 1–2, 1–3, 2–2, 2–3, 3–3. The total pressure profile, reported on the `PRESSUREPROFILE` line, will contain the internal contributions in the data sets corresponding to 1–1, 2–2, etc.

- pressureProfileAtomTypesFile < Atom type partition assignments >
  Acceptable Values: PDB file
  Default Value: coordinate file
  Description: If `pressureProfileAtomTypes` is greater than 1, NAMD will assign atoms to types based on the corresponding value in `pressureProfileAtomTypesCol`. The type for each atom must be strictly less than `pressureProfileAtomTypes`!

- pressureProfileAtomTypesCol < `pressureProfileAtomTypesFile` PDB column >
  Acceptable Values: PDB file
  Default Value: B
  Description:

Here is an example snippet from a NAMD input that can be used to compute the Ewald component of the pressure profile. It assumes that the coordinates were saved in the dcd file `pp03.dcd`) every 500 timesteps.

```
Pme              on
PmeGridSizeX     64
PmeGridSizeY     64
PmeGridSizeZ     64

exclude          scaled1-4
1-4scaling       1.0

switching on
switchdist       9
cutoff           10
pairlistdist     11

pressureProfile        on
pressureProfileSlabs    30
pressureProfileFreq     100
pressureProfileAtomTypes 6
pressureProfileAtomTypesFile atomtypes.pdb
pressureProfileEwald   on
pressureProfileEwaldX  16
pressureProfileEwaldY  16
pressureProfileEwaldZ  16

set ts 0
firstTimestep $ts

coorfile open dcd pp03.dcd
while { [coorfile read] != -1 } {
  incr ts 500
  firstTimestep $ts
  run 0
```

```
}
coorfile close
```

13 Translation between NAMD and X-PLOR configuration parameters

NAMD was designed to provide many of the same molecular dynamics functions that X-PLOR provides. As such, there are many similarities between the types of parameters that must be passed to both X-PLOR and NAMD. This section describes relations between similar NAMD and X-PLOR parameters.

- **NAMD Parameter:** `cutoff`
  X-PLOR Parameter: `CTOFNB`
  When full electrostatics are not in use within NAMD, these parameters have exactly the same meaning — the distance at which electrostatic and van der Waals forces are truncated. When full electrostatics are in use within NAMD, the meaning is still very similar. The van der Waals force is still truncated at the specified distance, and the electrostatic force is still computed at every timestep for interactions within the specified distance. However, the NAMD integration uses multiple time stepping to compute electrostatic force interactions beyond this distance every `stepspercycle` timesteps.

- **NAMD Parameter:** `vdwswitchdist`
  X-PLOR Parameter: `CTONNB`
  Distance at which the van der Waals switching function becomes active.

- **NAMD Parameter:** `pairlistdist`
  X-PLOR Parameter: `CUTNb`
  Distance within which interaction pairs will be included in pairlist.

- **NAMD Parameter:** `1-4scaling`
  X-PLOR Parameter: `E14Fac`
  Scaling factor for 1-4 pair electrostatic interactions.

- **NAMD Parameter:** `dielectric`
  X-PLOR Parameter: `EPS`
  Dielectric constant.

- **NAMD Parameter:** `exclude`
  X-PLOR Parameter: `NBXMod`
  Both parameters specify which atom pairs to exclude from non-bonded interactions. The ability to ignore explicit exclusions is *not* present within NAMD, thus only positive values of `NBXMod` have NAMD equivalents. These equivalences are

  - `NBXMod=1` is equivalent to `exclude=none` — no atom pairs excluded,
  - `NBXMod=2` is equivalent to `exclude=1-2` — only 1-2 pairs excluded,
  - `NBXMod=3` is equivalent to `exclude=1-3` — 1-2 and 1-3 pairs excluded,
  - `NBXMod=4` is equivalent to `exclude=1-4` — 1-2, 1-3, and 1-4 pairs excluded,
  - `NBXMod=5` is equivalent to `exclude=scaled1-4` — 1-2 and 1-3 pairs excluded, 1-4 pairs modified.

- **NAMD Parameter:** `switching`
  X-PLOR Parameter: `SHIFt, VSWItch, and TRUNcation`
  Activating the NAMD option `switching` is equivalent to using the X-PLOR options `SHIFt` and `VSWItch`. Deactivating `switching` is equivalent to using the X-PLOR option `TRUNcation`.

- **NAMD Parameter:** `temperature`
  X-PLOR Parameter: `FIRSttemp`
  Initial temperature for the system.

- **NAMD Parameter:** `rescaleFreq`
  X-PLOR Parameter: `IEQFrq`
  Number of timesteps between velocity rescaling.

- **NAMD Parameter:** `rescaleTemp`
  X-PLOR Parameter: `FINAltemp`
  Temperature to which velocities are rescaled.

- **NAMD Parameter:** `restartname`
  X-PLOR Parameter: `SAVE`
  Filename prefix for the restart files.

- **NAMD Parameter:** `restartfreq`
  X-PLOR Parameter: `ISVFrq`
  Number of timesteps between the generation of restart files.

- **NAMD Parameter:** `DCDfile`
  X-PLOR Parameter: `TRAJectory`
  Filename for the position trajectory file.

- **NAMD Parameter:** `DCDfreq`
  X-PLOR Parameter: `NSAVC`
  Number of timesteps between writing coordinates to the trajectory file.

- **NAMD Parameter:** `velDCDfile`
  X-PLOR Parameter: `VELOcity`
  Filename for the velocity trajectory file.

- **NAMD Parameter:** `velDCDfreq`
  X-PLOR Parameter: `NSAVV`
  Number of timesteps between writing velocities to the trajectory file.

- **NAMD Parameter:** `numsteps`
  X-PLOR Parameter: `NSTEp`
  Number of simulation timesteps to perform.

14 Sample configuration files

This section contains some simple example NAMD configuration files to serve as templates.

This file shows a simple configuration file for alanin. It performs basic dynamics with no output files or special features.

```
# protocol params
numsteps        1000

# initial config
coordinates     alanin.pdb
temperature     300K
seed            12345

# output params
outputname      /tmp/alanin
binaryoutput    no

# integrator params
timestep        1.0

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
1-4scaling      1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20
```

This file is again for alanin, but shows a slightly more complicated configuration. The system is periodic, a coordinate trajectory file and a set of restart files are produced.

```
# protocol params
numsteps        1000

# initial config
coordinates     alanin.pdb
temperature     300K
seed            12345

# periodic cell
cellBasisVector1   33.0 0 0
cellBasisVector2   0 32.0 0
cellBasisVector3   0 0 32.5

# output params
outputname      /tmp/alanin
binaryoutput    no
DCDfreq         10
restartfreq     100

# integrator params
timestep        1.0

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
1-4scaling      1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20
```

This file shows another simple configuration file for alanin, but this time with full electrostatics using PME and multiple timestepping.

```
# protocol params
numsteps        1000

# initial config
coordinates     alanin.pdb
temperature     300K
seed            12345

# periodic cell
cellBasisVector1   33.0 0 0
cellBasisVector2   0 32.0 0
cellBasisVector3   0 0 32.5

# output params
outputname      /tmp/alanin
binaryoutput    no
DCDfreq         10
restartfreq     100

# integrator params
timestep        1.0
fullElectFrequency  4

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
1-4scaling      1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20

# full electrostatics
PME             on
PMEGridSizeX    32
PMEGridSizeY    32
PMEGridSizeZ    32
```

This file demonstrates the analysis of a DCD trajectory file using NAMD. The file `pair.pdb` contains the definition of pair interaction groups; NAMD will compute the interaction energy and force between these groups for each frame in the DCD file. It is assumed that coordinate frames were written every 1000 timesteps. See Sec. 12.1 for more about pair interaction calculations.

```
# initial config
coordinates     alanin.pdb
temperature     0

# output params
outputname      /tmp/alanin-analyze
binaryoutput    no

# integrator params
timestep        1.0

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
1-4scaling      1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20

# Atoms in group 1 have a 1 in the B column; group 2 has a 2.
pairInteraction on
pairInteractionFile pair.pdb
pairInteractionCol B
pairInteractionGroup1 1
pairInteractionGroup2 2

# First frame saved was frame 1000.
set ts 1000


coorfile open dcd /tmp/alanin.dcd

# Read all frames until nonzero is returned.
while { ![coorfile read] } {
  # Set firstTimestep so our energy output has the correct TS.
  firstTimestep $ts
  # Compute energies and forces, but don't try to move the atoms.
  run 0
```

```
  incr ts 1000
}
coorfile close
```

15 Running NAMD

NAMD runs on a variety of serial and parallel platforms. While it is trivial to launch a serial program, a parallel program depends on a platform-specific library such as MPI to launch copies of itself on other nodes and to provide access to a high performance network such as Myrinet or InfiniBand if one is available.

For typical workstations (Windows, Linux, Mac OS X, or other Unix) with only ethernet networking (hopefully gigabit), NAMD uses the Charm++ native communications layer and the program charmrun to launch namd2 processes for parallel runs (either exclusively on the local machine with the ++local option or on other hosts as specified by a nodelist file). The namd2 binaries for these platforms can also be run directly (known as standalone mode) for single process runs.

For workstation clusters and other massively parallel machines with special high-performance networking, NAMD uses the system-provided MPI library (with a few exceptions) and standard system tools such as mpirun are used to launch jobs. Since MPI libraries are very often incompatible between versions, you will likely need to recompile NAMD and its underlying Charm++ libraries to use these machines in parallel (the provided non-MPI binaries should still work for serial runs.) The provided charmrun program for these platforms is only a script that attempts to translate charmrun options into mpirun options, but due to the diversity of MPI libraries it often fails to work.

15.1 Individual Windows, Linux, Mac OS X, or Other Unix Workstations

Individual workstations use the same version of NAMD as workstation networks, but running NAMD is much easier. If your machine has only one processor core you can run the any non-MPI namd2 binary directly:

```
namd2 <configfile>
```

For multicore workstations, Windows and Mac OX X (Intel) released binaries are based on "multicore" builds of Charm++ that can run multiple threads. These multicore builds lack a network layer, so they can only be used on a single machine. The Solaris (Sparc and x86-64) released binaries are based on "smp" builds of Charm++ that can be used with multiple threads on either a single machine like a multicore build, or across a network. For best performance use one thread per processor with the +p option:

```
namd2 +p<procs> <configfile>
```

For other multiprocessor workstations the included charmrun program is needed to run multiple namd2 processes. The ++local option is also required to specify that only the local machine is being used:

```
charmrun namd2 ++local +p<procs> <configfile>
```

You may need to specify the full path to the namd2 binary.

15.2   Linux or Other Unix Workstation Networks

The same binaries used for individual workstations as described above (other than pure "multicore" builds and MPI builds) can be used with charmrun to run in parallel on a workstation network. The only difference is that you must provide a "nodelist" file listing the machines where namd2 processes should run, for example:

```
group main
host brutus
host romeo
```

The "group main" line defines the default machine list. Hosts brutus and romeo are the two machines on which to run the simulation. Note that charmrun may run on one of those machines, or charmrun may run on a third machine. All machines used for a simulation must be of the same type and have access to the same namd2 binary.

By default, the "rsh" command is used to start namd2 on each node specified in the nodelist file. You can change this via the CONV_RSH environment variable, i.e., to use ssh instead of rsh run "setenv CONV_RSH ssh" or add it to your login or batch script. You must be able to connect to each node via rsh/ssh without typing your password; this can be accomplished via a .rhosts files in your home directory, by an /etc/hosts.equiv file installed by your sysadmin, or by a .ssh/authorized_keys file in your home directory. You should confirm that you can run "ssh hostname pwd" (or "rsh hostname pwd") without typing a password before running NAMD. Contact your local sysadmin if you have difficulty setting this up. If you are unable to use rsh or ssh, then add "setenv CONV_DAEMON" to your script and run charmd (or charmd_faceless, which produces a log file) on every node.

You should now be able to try running NAMD as:

```
charmrun namd2 +p<procs> <configfile>
```

If this fails or just hangs, try adding the ++verbose option to see more details of the startup process. You may need to specify the full path to the namd2 binary. Charmrun will start the number of processes specified by the +p option, cycling through the hosts in the nodelist file as many times as necessary. You may list multiprocessor machines multiple times in the nodelist file, once for each processor.

You may specify the nodelist file with the "++nodelist" option and the group (which defaults to "main") with the "++nodegroup" option. If you do not use "++nodelist" charmrun will first look for "nodelist" in your current directory and then ".nodelist" in your home directory.

Some automounters use a temporary mount directory which is prepended to the path returned by the pwd command. To run on multiple machines you must add a "++pathfix" option to your nodelist file. For example:

```
group main ++pathfix /tmp\_mnt /
host alpha1
host alpha2
```

There are many other options to charmrun and for the nodelist file. These are documented at in the Charm++ Installation and Usage Manual available at http://charm.cs.uiuc.edu/manuals/ and a list of available charmrun options is available by running charmrun without arguments.

If your workstation cluster is controlled by a queueing system you will need build a nodelist file in your job script. For example, if your queueing system provides a HOST_FILE environment variable:

```
set NODES = 'cat $HOST_FILE'
set NODELIST = $TMPDIR/namd2.nodelist
echo group main >! $NODELIST
foreach node ( $nodes )
  echo host $node >> $NODELIST
end
@ NUMPROCS = 2 * $#NODES
charmrun namd2 +p$NUMPROCS ++nodelist $NODELIST <configfile>
```

Note that NUMPROCS is twice the number of nodes in this example. This is the case for dual-processor machines. For single-processor machines you would not multiply $#NODES by two.

Note that these example scripts and the setenv command are for the csh or tcsh shells. They must be translated to work with sh or bash.

15.3 Windows Workstation Networks

These are no longer supported, but NAMD has been reported to compile on Windows HPC Server.

15.4 SGI Altix

Be sure that the MPI_DSM_DISTRIBUTE environment variable is set, then use the Linux-Itanium-MPI-Altix version of NAMD along with the system mpirun:

```
mpirun -np <procs> <configfile>
```

15.5 IBM POWER Clusters

Run the MPI version of NAMD as you would any POE program. The options and environment variables for poe are various and arcane, so you should consult your local documentation for recommended settings. As an example, to run on Blue Horizon one would specify:

```
poe namd2 <configfile> -nodes <procs/8> -tasks_per_node 8
```

15.6 Memory Usage

NAMD has traditionally used less than 100MB of memory even for systems of 100,000 atoms. With the reintroduction of pairlists in NAMD 2.5, however, memory usage for a 100,000 atom system with a 12A cutoff can approach 300MB, and will grow with the cube of the cutoff. This extra memory is distributed across processors during a parallel run, but a single workstation may run out of physical memory with a large system.

To avoid this, NAMD now provides a pairlistMinProcs config file option that specifies the minimum number of processors that a run must use before pairlists will be enabled (on fewer processors small local pairlists are generated and recycled rather than being saved, the default is "pairlistMinProcs 1"). This is a per-simulation rather than a compile time option because memory usage is molecule-dependent.

15.7 Improving Parallel Scaling

While NAMD is designed to be a scalable program, particularly for simulations of 100,000 atoms or more, at some point adding additional processors to a simulation will provide little or no extra performance. If you are lucky enough to have access to a parallel machine you should measure NAMD's parallel speedup for a variety of processor counts when running your particular simulation. The easiest and most accurate way to do this is to look at the "Benchmark time:" lines that are printed after 20 and 25 cycles (usually less than 500 steps). You can monitor performance during the entire simulation by adding "outputTiming *steps*" to your configuration file, but be careful to look at the "wall time" rather than "CPU time" fields on the "TIMING:" output lines produced. For an external measure of performance, you should run simulations of both 25 and 50 cycles (see the stepspercycle parameter) and base your estimate on the additional time needed for the longer simulation in order to exclude startup costs and allow for initial load balancing.

We provide both standard (UDP) and TCP based precompiled binaries for Linux clusters. We have observed that the TCP version is better on our dual processor clusters with gigabit ethernet while the basic UDP version is superior on our single processor fast ethernet cluster. When using the UDP version with gigabit you can add the +giga option to adjust several tuning parameters. Additional performance may be gained by building NAMD against an SMP version of Charm++ such as net-linux-smp or net-linux-smp-icc. This will use a communication thread for each process to respond to network activity more rapidly. For dual processor clusters we have found it that running two separate processes per node, each with its own communication thread, is faster than using the charmrun ++ppn option to run multiple worker threads. However, we have observed that when running on a single hyperthreaded processor (i.e., a newer Pentium 4) there is an additional 15% boost from running standalone with two threads (namd2 +p2) beyond running two processors (charmrun namd2 ++local +p2). For a cluster of single processor hyperthreaded machines an SMP version should provide very good scaling running one process per node since the communication thread can run very efficiently on the second virtual processor. We are unable to ship an SMP build for Linux due to portability problems with the Linux pthreads implementation needed by Charm++.

Extremely short cycle lengths (less than 10 steps) will also limit parallel scaling, since the atom migration at the end of each cycle sends many more messages than a normal force evaluation. Increasing pairlistdist from, e.g., cutoff + 1.5 to cutoff + 2.5, while also doubling stepspercycle from 10 to 20, may increase parallel scaling, but it is important to measure. When increasing stepspercycle, also try increasing pairlistspercycle by the same proportion.

NAMD should scale very well when the number of patches (multiply the dimensions of the patch grid) is larger or rougly the same as the number of processors. If this is not the case, it may be possible to improve scaling by adding "twoAwayX yes" to the config file, which roughly doubles the number of patches. (Similar options twoAwayY and twoAwayZ also exist, and may be used in combination, but this greatly increases the number of compute objects. twoAwayX has the unique advantage of also improving the scalability of PME.)

Additional performance tuning suggestions and options are described at http://www.ks.uiuc.edu/Research/namd/wiki/?NamdPerformanceTuning

16 NAMD Availability and Installation

NAMD is distributed freely for non-profit use. NAMD 2.7b1 is based on the Charm++ messaging system and the Converse communication layer (`http://charm.cs.uiuc.edu/`) which have been ported to a wide variety of parallel platforms. This section describes how to obtain and install NAMD 2.7b1.

16.1 How to obtain NAMD

NAMD may be downloaded from `http://www.ks.uiuc.edu/Research/namd/`. You will be required to provide minimal registration information and agree to a license before receiving access to the software. Both source and binary distributions are available.

16.2 Platforms on which NAMD will currently run

NAMD should be portable to any parallel platform with a modern C++ compiler to which Charm and Converse have been ported. Precompiled NAMD 2.7b1 binaries are available for download for the following platforms:

- Windows (XP, etc.) on x86 processors

- Mac OS X on PowerPC and Intel processors

- Linux on x86, x86-64, and Itanium processors

- SGI Altix (with MPI)

- AIX on POWER processors (with and without MPI)

- Solaris on Sparc and x86-64 processors

NAMD has been ported to the Cray XT for both Catamount and compute-node Linux, but the lack of dynamic linking on this platform often results in the need to recompile NAMD after system upgrades so we are unable to distribute useful binaries.

NAMD has been ported to the IBM Blue Gene L and P, but we do not have regular access to these machines. Binaries provided by IBM may be available for download from our website.

16.3 Compiling NAMD

We provide complete and optimized binaries for all platforms to which NAMD has been ported. It should not be necessary for you to compile NAMD unless you wish to add or modify features or to improve performance by using an MPI library that takes advantage of special networking hardware.

Directions for compiling NAMD are contained in the release notes, which are available from the NAMD web site `http://www.ks.uiuc.edu/Research/namd/` and are included in all distributions.

16.4 Documentation

All available NAMD documentation is available for download without registration via the NAMD web site `http://www.ks.uiuc.edu/Research/namd/`.

References

[1] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, New York, 1987.

[2] P. H. Axelsen and D. Li. Improved convergence in dual–topology free energy calculations through use of harmonic restraints. *J. Comput. Chem.*, 19:1278–1283, 1998.

[3] C. H. Bennett. Efficient estimation of free energy differences with monte carlo data. *J. Comp. Phys.*, 22:245–268, 1976.

[4] F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, J. E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The protein data bank: A computer-based archival file for macromolecular structures. *J. Mol. Biol.*, 112:535–542, 1977.

[5] T. C. Beutler, A. E. Mark, R. C. van Schaik, P. R. Gerber, and W. F. van Gunsteren. Avoiding singularities and numerical instabilities in free energy calculations based on molecular simulations. *Chem. Phys. Lett.*, 222:529–539, 1994.

[6] D. L. Beveridge and F. M. DiCapua. Free energy via molecular simulation: Applications to chemical and biomolecular systems. *Annu. Rev. Biophys. Biophys.*, 18:431–492, 1989.

[7] S. Boresch and M. Karplus. The role of bonded terms in free energy simulations: I. theoretical analysis. *J. Phys. Chem. A*, 103:103–118, 1999.

[8] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: a program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem.*, 4(2):187–217, 1983.

[9] A. T. Brünger. *X-PLOR, Version 3.1, A System for X-ray Crystallography and NMR*. The Howard Hughes Medical Institute and Department of Molecular Biophysics and Biochemistry, Yale University, 1992.

[10] G. Bussi, A. Laio, and M. Parrinello. Equilibrium free energies from nonequilibrium metadynamics. *Phys. Rev. Lett.*, 96(9):090601, 2006.

[11] A. Carter, E, G. Ciccotti, J. T. Hynes, and R. Kapral. Constrained reaction coordinate dynamics for the simulation of rare events. *Chem. Phys. Lett.*, 156:472–477, 1989.

[12] C. Chipot and D. A. Pearlman. Free energy calculations. the long and winding gilded road. *Mol. Sim.*, 28:1–12, 2002.

[13] C. Chipot and A. Pohorille, editors. *Free energy calculations. Theory and applications in chemistry and biology.* Springer Verlag, 2007.

[14] G. Ciccotti, R. Kapral, and E. Vanden-Eijnden. Blue moon sampling, vectorial reaction coordinates, and unbiased constrained dynamics. *ChemPhysChem*, 6(9):1809–1814, 2005.

[15] E. A. Coutsias, C. Seok, and K. A. Dill. Using quaternions to calculate RMSD. *J. Comput. Chem.*, 25(15):1849–1857, 2004.

[16] E. Darve and A. Pohorille. Calculating free energies using average force. *J. Chem. Phys,*, 115(20):9169–9183, NOV 22 2001.

[17] E. Darve, D. Rodríguez-Gómez, and A. Pohorille. Adaptive biasing force method for scalar and vector free energy calculations. *J. Chem. Phys.*, 128(14):144120, 2008.

[18] W. K. den Otter. Thermodynamic integration of the free energy along a reaction coordinate in cartesian coordinates. *J. Chem. Phys.*, 112:7283–7292, 2000.

[19] D. Frenkel and B. Smit. *Understanding Molecular Simulation From Algorithms to Applications*. Academic Press, California, 2002.

[20] J. Gao, K. Kuczera, B. Tidor, and M. Karplus. Hidden thermodynamics of mutant proteins: A molecular dynamics analysis. *Science*, 244:1069–1072, 1989.

[21] M. K. Gilson, J. A. Given, B. L. Bush, and J. A. McCammon. The statistical–thermodynamic basis for computation of binding affinities: A critical review. *Biophys. J.*, 72:1047–1069, 1997.

[22] H. Grubmüller. Predicting slow structural transitions in macromolecular systems: Conformational flooding. *Phys. Rev. E*, 52(3):2893–2906, Sep 1995.

[23] J. Hénin and C. Chipot. Overcoming free energy barriers using unconstrained molecular dynamics simulations. *J. Chem. Phys.*, 121:2904–2914, 2004.

[24] T. Huber, A. E. Torda, and W. van Gunsteren. Local elevation - A method for improving the searching properties of molecular-dynamics simulation. *Journal of Computer-Aided Molecular Design*, 8(6):695–708, DEC 1994.

[25] G. Hummer and I. Kevrekidis. Coarse molecular dynamics of a peptide fragment: Free energy, kinetics, and long-time dynamics computations. *Journal of Chemical Physics*, 118(23):10762 – 10773, JUN 15 2003.

[26] W. Humphrey and A. Dalke. VMD user guide (Version 0.94). Beckman Institute Technical Report TB-94-07, University of Illinois, 1994.

[27] M. Iannuzzi, A. Laio, and M. Parrinello. Efficient exploration of reactive potential energy surfaces using car-parrinello molecular dynamics. *Phys. Rev. Lett.*, 90(23):238302, 2003.

[28] P. M. King. Free energy via molecular simulation: A primer. In W. F. Van Gunsteren, P. K. Weiner, and A. J. Wilkinson, editors, *Computer simulation of biomolecular systems: Theoretical and experimental applications*, volume 2, pages 267–314. ESCOM, Leiden, 1993.

[29] J. G. Kirkwood. Statistical mechanics of fluid mixtures. *J. Chem. Phys.*, 3:300–313, 1935.

[30] P. A. Kollman. Free energy calculations: Applications to chemical and biochemical phenomena. *Chem. Rev.*, 93:2395–2417, 1993.

[31] A. Laio and M. Parrinello. Escaping free-energy minima. *Proc. Natl. Acad. Sci. USA*, 99(20):12562–12566, 2002.

[32] N. Lu, D. A. Kofke, and T. B. Woolf. Improving the efficiency and reliability of free energy perturbation calculations using overlap sampling methods. *J. Comput. Chem.*, 25:28–39, 2004.

[33] Z. M., T. P. Straatsma, and M. J. A. Separation-shifted scaling, a new scaling method for Lennard-Jones interactions in thermodynamic integration. *J. Chem. Phys.*, 100:9025–9031, 1994.

[34] A. E. Mark. Free energy perturbation calculations. In P. v. R. Schleyer, N. L. Allinger, T. Clark, J. Gasteiger, P. A. Kollman, H. F. Schaefer III, and P. R. Schreiner, editors, *Encyclopedia of computational chemistry*, volume 2, pages 1070–1083. Wiley and Sons, Chichester, 1998.

[35] J. A. McCammon and S. C. Harvey. *Dynamics of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, 1987.

[36] D. A. Pearlman. A comparison of alternative approaches to free energy calculations. *J. Phys. Chem.*, 98:1487–1493, 1994.

[37] P. Raiteri, A. Laio, F. L. Gervasio, C. Micheletti, and M. Parrinello. Efficient reconstruction of complex free energy landscapes by multiple walkers metadynamics. *J. Phys. Chem. B*, 110(8):3533–9, 2005.

[38] A. Roitberg and R. Elber. Modeling side chains in peptides and proteins: Application of the locally enhanced sampling technique and the simulated annealing methods to find minimum energy conformations. *J. Chem. Phys.*, 95:9277–9287, 1991.

[39] M. J. Ruiz-Montero, D. Frenkel, and J. J. Brey. Efficient schemes to compute diffusive barrier crossing rates. *Mol. Phys.*, 90:925–941, 1997.

[40] C. Simmerling, T. Fox, and P. A. Kollman. Use of locally enhanced sampling in free energy calculations: Testing and application to the $\alpha \rightarrow \beta$ anomerization of glucose. *J. Am. Chem. Soc.*, 120(23):5771–5782, 1998.

[41] C. Simmerling, M. R. Lee, A. R. Ortiz, A. Kolinski, J. Skolnick, and P. A. Kollman. Combining MONSSTER and LES/PME to predict protein structure from amino acid sequence: Application to the small protein CMTI-1. *J. Am. Chem. Soc.*, 122(35):8392–8402, 2000.

[42] T. P. Straatsma and J. A. McCammon. Multiconfiguration thermodynamic integration. *J. Chem. Phys.*, 95:1175–1118, 1991.

[43] T. P. Straatsma and J. A. McCammon. Computational alchemy. *Annu. Rev. Phys. Chem.*, 43:407–435, 1992.

[44] W. F. van Gunsteren. Methods for calculation of free energies and binding constants: Successes and problems. In W. F. Van Gunsteren and P. K. Weiner, editors, *Computer simulation of biomolecular systems: Theoretical and experimental applications*, pages 27–59. Escom, The Netherlands, 1989.

[45] A. F. Voter. Hyperdynamics: Accelerated molecular dynamics of infrequent events. *Phys. Rev. Lett.*, 78(20):3908–3911, May 1997.

[46] R. W. Zwanzig. High–temperature equation of state by a perturbation method. i. nonpolar gases. *J. Chem. Phys.*, 22:1420–1426, 1954.

Index