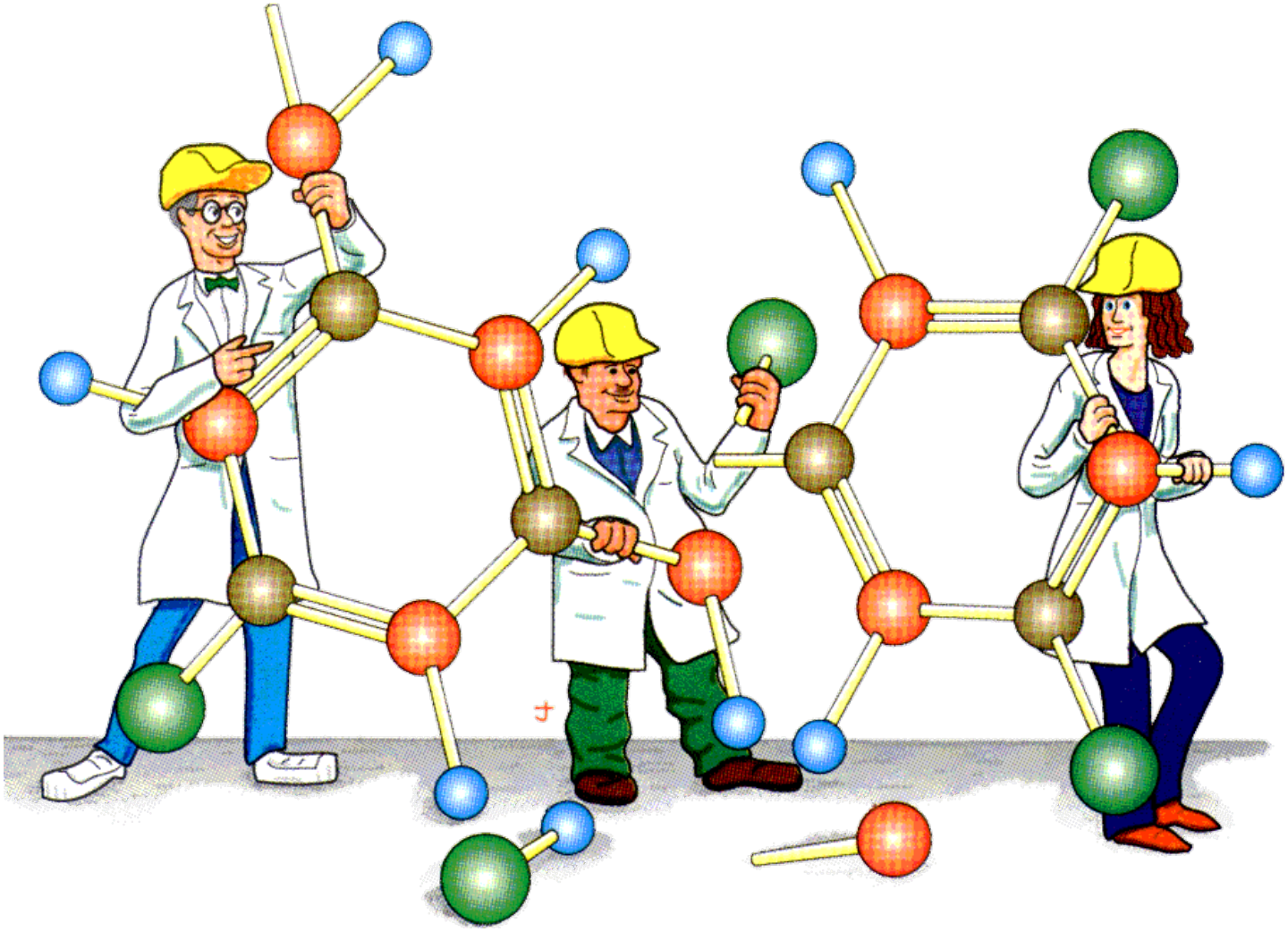
TINKER

Tools for Molecular Design



Version 8.8 June 2020

User's Guide Cover Illustration by Jay Nelson
Courtesy of Prof. R. T. Paine, Univ. of New Mexico

V8.8 6/20

Tinker User's Guide

TinkerTools Organization

May 23, 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION TO THE SOFTWARE

1.1 What is Tinker?

Welcome to the Tinker molecular modeling package! Tinker is designed to be an easily used and flexible system of programs and routines for molecular mechanics and dynamics as well as other energy-based and structural manipulation calculations. It is intended to be modular enough to enable development of new computational methods and efficient enough to meet most production calculation needs. Rather than incorporating all the functionality in one monolithic program, Tinker provides a set of relatively small programs that interoperate to perform complex computations. New programs can be easily added by modelers with only limited programming experience.

1.2 Features and Capabilities

The series of major programs included in the distribution system perform the following core tasks:

(1) building protein and nucleic acid models from sequence

(2) energy minimization and structural optimization

(3) analysis of energy distribution within a structure

(4) molecular dynamics and stochastic dynamics

(5) simulated annealing with a choice of cooling schedules

(6) normal modes and vibrational frequencies

(7) conformational search and global optimization

(8) transition state location and conformational pathways

(9) fitting of energy parameters to crystal data

(10) distance geometry with pairwise metrization

(11) molecular volumes and surface areas

(12) free energy changes for structural mutations

(13) advanced algorithms based on potential smoothing

Many of the various energy minimization and molecular dynamics computations can be performed on full or partial structures, over Cartesian, internal or rigid body coordinates, and including a variety of boundary conditions and crystal cell types. Other programs are available to generate timing data and allow checking of potential function derivatives for coding errors. Special features are available to facilitate input and output of protein and nucleic acid structures. However, the basic core routines have no knowledge of biopolymer structure and can be used for general molecular systems.

Due to its emphasis on ease of modification, Tinker differs from many other currently available molecular modeling packages in that the user is expected to be willing to write simple "front-end" programs and make some alterations at the source code level. The main programs provided should be considered as templates for the users to change according to their wishes. All subroutines are internally documented and structured programming practices are adhered to throughout. The result, it is hoped, will be a calculational system which can be tailored to local needs and desires.

The core Tinker system consists of over 240,000 lines of source written entirely in a portable Fortran superset. Use is made of only some very common extensions that aid in writing highly structured code. The current version of the package has been ported to a wide range of computers with no or extremely minimal changes. Tested systems include: Ubuntu, CentOS and Red Hat Linux, Microsoft Windows 10 and earlier, Apple MacOS, and various older Unix-based workstations under vendor supplied Unix. At present, our new code is written on various Linux platforms, and occasionally tested for compatibility on various of the other machine and OS combinations listed above. At present, our primary source code development efforts are in Fortran, using a portable subset of Fortran90 with some common extensions. A machine-translated C version of Tinker is currently available, and a hand-translated optimized C version of a previous Tinker release is available for inspection. Conversion to C or C++ is under consideration, but not being actively pursued at this time.

The basic design of the energy function engine used by the Tinker system allows usage of several different parameter sets. At present we are distributing parameters that implement several Amber and CHARMM potentials, MM2, MM3, OPLS-UA, OPLS-AA, MMFF, Liam Dang's polarizable potentials, and our own AMOEBA (Atomic Multipole Optimized Energetics for Biomolecular Applications), AMOEBA+, and HIPPO (Hydrogen-like Intermolecular Polarizable Potential) force fields. In most cases, the source code separates the geometric manipulations needed for energy derivatives from the actual form of the energy function itself. Several other literature parameter sets are being considered for possible future development (later versions of CHARMM and Amber, as well as GROMOS, ENCAD, MM4, UFF, etc.), and many of the alternative potential function forms reported in the literature can be implemented directly or after minor code changes.

Much of the software in the Tinker package has been heavily used and well tested, but some modules are still in a fairly early stage of development. Further work on the Tinker system is planned in three main areas: (1) extension and improvement of the potential energy parameters including additional parameterization and testing of our polarizable multipole AMOEBA force field, (2) coding of new computational algorithms including additional methods for free energy determination, torsional Monte Carlo and molecular dynamics sampling, advanced methods for long range interactions, better transition state location, and further application of the potential smoothing paradigm, and (3) further development of Force Field Explorer, a Java-based GUI front-end to the Tinker programs that provides for calculation setup, launch and control as well as basic molecular visualization.

1.3 Contact Information

Questions and comments regarding the Tinker package, including suggestions for improvements and changes should be made to the author:

Professor Jay William Ponder Department of Chemistry, Box 1134 Washington University in Saint Louis One Brookings Hall Saint Louis, MO 63130 U.S.A.

office: Louderman Hall, Room 453 phone: (314) 935-4275 fax: (314) 935-4481 email: ponder@dasher.wustl.edu

In addition, an Internet web site containing an online version of this User's Guide, the most recent distribution version of the full Tinker package and other useful information can be found at https://dasher.wustl.edu/tinker/, the Home Page for the Tinker Molecular Modeling Package. Tinker and related software packages are also available from GitHub at the site https://github.com/TinkerTools/Tinker.git/.

CHAPTER

TWO

INSTALLATION ON YOUR COMPUTER

2.1 How to Obtain a Copy of Tinker

The Tinker package is distributed on the Internet at the Ponder lab's Tinker web site located at https://dasher.wustl.edu/tinker/, or via download from the Github site for the TinkerTools organization at https://github.com/TinkerTools/Tinker/. After unpacking the distribution, you can build a set of Tinker executables on almost any machine with a Fortran compiler. Makefiles, a CMakeLists.txt file for cmake, as well as standalone scripts to compile, build object libraries, and link executables on a wide variety of machine-CPU-operating system combinations are provided.

2.2 Prebuilt Tinker Executables

Pre-built Tinker executables for Linux, MacOS, and Windows are also available for download from the sites mentioned above. They should run on most recent vintage machines using the above operating systems, and can handle a maximum of 1 million atoms provided sufficient memory is available. The Linux executables require at least glibc-2.6 or later. Note starting with Tinker 8, we no longer provide pre-built executables for any 32-bit operating systems.

The provided executables are OpenMP capable, but do not support APBS or the Tinker-FFE interface. You will still need to have a copy of the complete Tinker distribution as it contains the parameter sets, examples, benchmarks, test files and documentation required to use the package.

2.3 Building your Own Executables

The compilation and building of the Tinker executables should be easy for most of the common Linux, MacOS and Windows computers. We provide in the /make area of the distribution a Makefile that with minor modification can be used to build Tinker on any of these machines. As an alternative to Makefiles, we also provide machine-specific directories with three separate shell scripts to compile the source, build an object library, and link binary executables.

The first step in building Tinker using the script files is to run the appropriate compile.make script for your operating system and compiler version. Next you must use a library.make script to create an archive of object code modules. Finally, run a link.make script to produce the complete set of Tinker executables. The executables can be renamed and moved to wherever you like by editing

and running the "rename" script. These steps will produce executables that can run from the command line, but without the capability to interact with the FFE GUI. Building FFE-enabled Tinker executables involves replacing the sockets.f source file with sockets.c, and included the object from the C code in the Tinker object library. Then executables must be linked against Java libraries in addition to the usual resources. Sample compgui.make and linkgui.make scripts are provided for systems capable of building GUI-enabled executables.

Regardless of your target machine, only a few small pieces of code can possibly require attention prior to building. The most common source alterations are to the master array dimensions found in the source file sizes.f. The basic limit is on the number of atoms allowed, "maxatm". This parameter can be set to 1000000 or more on most workstations. Personal computers with minimal memory may need a lower limit, depending on available memory, swap space and other resources. A description of the other parameter values is contained in the header of the file.

2.4 Tinker-FFE (Force Field Explorer)

Tinker-FFE, formerly Force Field Explorer, is a Java-based GUI for the Tinker package. It provides visualization for Tinker molecule files, as well as launching of Tinker calculations from a graphical interface. Tinker-FFE for Linux, MacOS and Windows can be downloaded from the Ponder lab Tinker web site as "installation kits" containing the FFE GUI and an FFE-enabled version of Tinker. Tinker-FFE requires a 64-bit CPU and operating system, as 32-bit systems are no longer supported.

Integration with Tinker, including the ability to interactively run Tinker calculations, and to access molecule downloads from the PubChem, NCI and PDB databases make Tinker-FFE a useful tool in classroom teaching environments. For research work, we recommend using the latest command line version of Tinker for numerical calculations, and using FFE or another visualization program to view results. Several other visualization programs (including VMD, Avogadro, Jmol, MOLDEN, WebMO, some PyMOL versions, etc.) can display Tinker structure and MD trajectory files.

The Tinker-FFE Installer for Linux is provided as a gzipped shell script. Uncompress the the .gz archive to produce an .sh script, and then run the script. The script must have the "executable" attribute, set via "chmod +x installer-file-name.sh", prior to being run.

The Tinker-FFE Installer for MacOS is provided as a .dmg disk image file. Double-click on the file to run the installer. MacOS 10.8 and later contains a security feature called Gatekeeper that keeps applications not obtained via the App Store or Apple-approved developers from being opened. Gatekeeper is enabled by default, and may result in the (incorrect!) error message: "Tinker-FFE Installer.app is damaged and can't be opened." To turn off Gatekeeper, go to the panel System Preferences > Security & Privacy > General, and set "Allow apps downloaded from:" to "Anywhere". This will require an Administrator account, and must be done before invoking the FFE installer. Once FFE is installed and launched for the first time, you can return the System Preference to its prior value. On Sierra (10.12) and later, the "Anywhere" option has been removed. In most cases the Security & Privacy panel will open and permit the user to run the installer. Alternatively, the "Anywhere" option can be restored by running the command "sudo spctl –master-disable" in a Terminal window.

The Tinker-FFE Installer for Windows is provided as a zipped executable. First, unzip the .zip file, then run the resulting executable .exe file. In order to perform minimizations or molecular dynamics from within FFE, some environment variables and symbolic links must be set prior to

using the program. A batch file named "FFESetupWin.bat" is installed in the main Tinker-FFE directory, which by default resides in the user's home directory. To complete the setup of FFE, this batch file should be run from a Command Prompt window following installation. It is only necessary to invoke this batch file once, as the settings should persist between logins.

For those wishing to modify the FFE GUI or build a version from source, we provide a complete development package for Tinker-FFE. This is a large download which contains the code for all components, including the Java source for FFE itself and the many required Java libraries. This package allows building Tinker-FFE on all three supported operating systems from a common code base. External requirements are the GNU compiler suite with gcc, g++ and gfortran (on Windows use MinGW-w64 compilers under Cygwin), and the Install4j Java installer builder. Note Install4j is a commercial product; only the compiler is needed, not the full Install4j GUI interface.

2.5 Documentation and Other Information

The documentation for the Tinker programs, including the guide you are currently reading, is located in the /doc subdirectory of the distribution. The documentation was prepared using the Sphinx documentation generator. Portable versions of the documentation are provided as PDF files and in HTML format for web display. Please read and return by mail the Tinker license. In particular, we note that Tinker is not an Open Source package as users are prohibited from redistribution of original or modified Tinker source code or binaries to other parties. While our intent is to distribute the Tinker code to anyone who wants it, the Ponder Lab would keep track of researchers using the package. The returned license forms also help us justify further development of Tinker. When new modules and capabilities become available, and when the inevitable bugs are uncovered, we will attempt to notify those who have returned a license form. Finally, we remind you that this software is copyrighted, and ask that it not be redistributed in any form.

2.6 Where to Direct Questions

Specific questions about the building or use of the Tinker package should be directed to ponder@dasher.wustl.edu. Tinker related questions or comments of more general interest can be posted on Twitter @TINKERtoolsMD. The Tinker developers monitor this account and will respond to the site or the individual poster as appropriate.

CHAPTER

THREE

TYPES OF INPUT & OUTPUT FILES

This section describes the basic file types used by the Tinker package. Let's say you wish to perform a calculation on a particular small organic molecule. Assume that the file name chosen for our input and output files is sample. Then all of the Tinker files will reside on the computer under the name sample.xxx where .xxx is any of the several extension types to be described below.

SAMPLE.XYZ

The .xyz file is the basic Tinker Cartesian coordinates file type. It contains a title line followed by one line for each atom in the structure. Each line contains: the sequential number within the structure, an atomic symbol or name, X-, Y-, and Z-coordinates, the force field atom type number of the atom, and a list of the atoms connected to the current atom. Except for programs whose basic operation is in torsional space, all Tinker calculations are done from some version of the .xyz format.

SAMPLE.INT

The .int file contains an internal coordinates representation of the molecular structure. It consists of a title line followed by one line for each atom in the structure. Each line contains: the sequential number within the structure, an atomic symbol or name, the force field atom type number of the atom, and internal coordinates in the usual Z-matrix format. For each atom the internal coordinates consist of a distance to some previously defined atom, and either two bond angles or a bond angle and a dihedral angle to previous atoms. The length, angle and dihedral definitions do not have to represent real bonded interactions. Following the last atom definition are two optional blank line separated sets of atom number pairs. The first list contains pairs of atoms that are covalently bonded, but whose bond length was not used as part of the atom definitions. These pairs are typically used to close ring structures. The second list contains ``bonds" that are to be broken, i.e., pairs of atoms that are not covalently bonded, but which were used to define a distance in the atom definitions.

SAMPLE.KEY

The keyword parameter file always has the extension .key and is optionally present during Tinker calculations. It contains values for any of a wide variety of switches and parameters that are used to change the course of the computation from the default. The detailed contents of this file is explained in a latter section of this User's Guide. If a molecular system specific keyfile, in this case sample.key, is not present, the the Tinker program will look in the same directory for a generic file named Tinker.key.

SAMPLE.DYN

The .dyn file contains values needed to restart a molecular or stochastic dynamics computation. It stores the current position, current velocity and current and previous accelerations for each atom, as well as the size and shape of any periodic box or crystal unit cell. This information can be used to start a new dynamics run from the final state of a previous run. Upon startup, the dynamics programs always check for the presence of a .dyn file and make use of it whenever possible. The .dyn file is updated concurrent with the saving of a new dynamics trajectory snapshot.

SAMPLE.END

The .end file type provides a mechanism to gracefully stop a running Tinker calculation. At appropriate checkpoints during a calculation, Tinker will test for the presence of a sample.end file, and if found will terminate the calculation after updating the output. The .end file can be created at any time during a computation, and will be detected when the next checkpoint is reached. The file may be of zero size, and its contents are unimportant. In the current version of Tinker, the .end mechanism is only available within dynamics-based programs.

SAMPLE.001, SAMPLE.002, ...

Several types of computations produce files containing a three or more digit extension (.001 as shown; or .002, .137, .5678, etc.). These are referred to as cycle files, and are used to store various types of output structures. The cycle files from a given computation are identical in internal structure to either the .xyz or .int files described above. For example, the vibrational analysis program can save the tenth normal mode in sample.010. A molecular dynamics-based program might save its tenth 0.1 picosecond frame (or an energy minimizer its tenth partially minimized intermediate) in a file of the same name.

SAMPLE.LOG

The Force Field Explorer interface to Tinker saves results of all calculations launched from the GUI to a log file with the .log suffix. Any output that would normally be directed to the screen after starting a program from the command line is appended to this log file by Force Field Explorer.

SAMPLE.ARC

A Tinker archive file is simply a series of .xyz Cartesian coordinate files appended together one after another. This file can be used to condense the results from intermediate stages of an optimization, frames from a molecular dynamics trajectory, or set of normal mode vibrations into a single file for storage. Tinker archive files can be displayed as sequential frame “movies” by the Force Field Explorer modeling program.

SAMPLE.PDB

This file type contains coordinate information in the PDB format developed by the Brookhaven Protein Data Bank for deposition of model structures based on macromolecular X-ray diffraction and NMR data. Although Tinker itself does not use .pdb files directly for input/output, auxiliary programs are provided with the system for interconverting .pdb files with the .xyz format described above.

SAMPLE.SEQ

This file type contains the primary sequence of a biopolymer in the standard one-letter code with 50 residues per line. The .seq file for a biopolymer is generated automatically when a PDB file is converted to Tinker .xyz format or when using the PROTEIN or NUCLEIC programs to build

a structure from sequence It is required for the reverse conversion of a Tinker file back to PDB format..

SAMPLE.FRAC

The fractional coordinates corresponding to the asymmetric unit of a crystal unit cell are stored in the .frac file. The internal format of this file is identical to the .xyz file; except that the coordinates are fractional instead of in Angstrom units.

SAMPLE.MOL2

File conversion to and from the Tripos Sybyl MOL2 file format is supported by Tinker. The utility programs XYZMOL2 and MOL2XYZ transform a Tinker XYZ file to MOL2 format, and the reverse.

PARAMETER FILES (*.PRM)

The potential energy parameter files distributed with the Tinker package all end in the extension .prm, although this is not required by the programs themselves. Each of these files contains a definition of the potential energy functional forms for that force field as well as values for individual energy parameters. For example, the mm3pro.prm file contains the energy parameters and definitions needed for a protein-specific version of the MM3 force field.

CHAPTER

FOUR

POTENTIAL ENERGY PROGRAMS

This section of the manual contains a brief description of each of the Tinker potential energy programs. A detailed example showing how to run each program is included in a later section. The programs listed below are all part of the main, supported distribution. Additional source code for various unsupported programs can be found in the /other directory of the Tinker distribution.

ALCHEMY

A simple program to perform very basic free energy perturbation calculations. This program is provided mostly for demonstration purposes. For example, we use ALCHEMY in a molecular modeling course laboratory exercise to perform such classic mutations as chloride to bromide and ethane to methanol in water. The present version uses the perturbation formula and windowing with an explicit mapping of atoms involved in the mutation ("Amber"-style), instead of thermodynamic integration and independent freely propagating groups of mutated atoms ("CHARMM"-style). Some of the code specific to this program is limited to the Amber and OPLS potential functional forms, but could be easily generalized to handle other potentials. A more general and sophisticated version is currently under development.

ANALYZE

Provides information about a specific molecular structure. The program will ask for the name of a structure file, which must be in the Tinker XYZ file format, and the type of analysis desired. Options allow output of: (1) total potential energy of the system, (2) breakdown of the energy by potential function type or over individual atoms, (3) computation of the total dipole moment and its components, moments of inertia and radius of gyration, (4) listing of the parameters used to compute selected interaction energies, (5) energies associated with specified individual interactions.

ANNEAL

Performs a molecular dynamics simulated annealing computation. The program starts from a specified input molecular structure in Tinker XYZ format. The trajectory is updated using either a modified Beeman or a velocity Verlet integration method. The annealing protocol is implemented by allowing smooth changes between starting and final values of the system temperature via the Groningen method of coupling to an external bath. The scaling can be linear or sigmoidal in nature. In addition, parameters such as cutoff distance can be transformed along with the temperature. The user must input the desired number of dynamics steps for both the equilibration and cooling phases, a time interval for the dynamics steps, and an interval between coordinate/trajectory saves. All saved coordinate sets along the trajectory are placed in sequentially numbered cycle files.

DYNAMIC

Performs a molecular dynamics (MD) or stochastic dynamics (SD) computation. Starts either from a specified input molecular structure (an XYZ file) or from a structure-velocity-acceleration set saved from a previous dynamics trajectory (a restart from a DYN file). MD trajectories are propagated using either a modified Beeman or a velocity Verlet integration method. SD is implemented via our own derivation of a velocity Verlet-based algorithm. In addition the program can perform full crystal calculations, and can operate in constant energy mode or with maintenance of a desired temperature and/or pressure using the Berendsen method of coupling to external baths. The user must input the desired number of dynamics steps, a time interval for the dynamics steps, and an interval between coordinate/trajectory saves. Coordinate sets along the trajectory can be saved as sequentially numbered cycle files or directly to a Tinker archive (ARC) file. At the same time that a point along the trajectory is saved, the complete information needed to restart the trajectory from that point is updated and stored in the DYN file.

GDA

A program to implement Straub's Gaussian Density Annealing algorithm over an effective series of analytically smoothed potential energy surfaces. This method can be viewed as an extended stochastic version of the diffusion equation method of Scheraga, et al., and also has many similar features to the Tinker Potential Smoothing and Search (PSS) series of programs. The current version of GDA is similar to but does not exactly reproduce Straub's published method and is limited to argon clusters and other simple systems involving only van der Waals interactions; further modification and development of this code is currently underway in the Ponder research group. As with other programs involving potential smoothing, GDA currently requires use of the smooth.prm force field parameters.

MINIMIZE

The MINIMIZE program performs a limited memory L-BFGS minimization of an input structure over Cartesian coordinates using a modified version of the algorithm of Jorge Nocedal. The method requires only the potential energy and gradient at each step along the minimization pathway. It requires storage space proportional to the number of atoms in the structure. The MINIMIZE procedure is recommended for preliminary minimization of trial structures to an RMS gradient of 1.0 to 0.1 kcal/mole/Ang. It has a relatively fast cycle time and is tolerant of poor initial structures, but converges in a slow, linear fashion near the minimum. The user supplies the name of the Tinker XYZ coordinates file and a target rms gradient value at which the minimization will terminate. Output consists of minimization statistics written to the screen or redirected to an output file, and the new coordinates written to updated XYZ files or to cycle files.

MINIROT

The MINIROT program uses the same limited memory L-BFGS method as MINIMIZE, but performs the computation in terms of dihedral angles instead of Cartesian coordinates. Output is saved in an updated .int file or in cycle files.

MINRIGID

The MINRIGID program is similar to MINIMIZE except that it operates on rigid bodies starting from a Tinker XYZ coordinate file and the rigid body group definitions found in the corresponding KEY file. Output is saved in an updated XYZ file or in cycle files.

MONTE

The MONTE program implements the Monte Carlo Minimization algorithm developed by Harold

Scheraga's group and others. The procedure takes Monte Carlo steps for either a single atom or a single torsional angle, then performs a minimization before application of the Metropolis sampling method. This results in effective sampling of a modified potential surface where the only possible energy levels are those of local minima on the original surface. The program can be easily modified to elaborate on the available move set.

NEWTON

A truncated Newton minimization method which requires potential energy, gradient and Hessian information. This procedure has significant advantages over standard Newton methods, and is able to minimize very large structures completely. Several options are provided with respect to minimization method and preconditioning of the Newton equations. The default options are recommended unless the user is familiar with the math involved. This program operates in Cartesian coordinate space and is fairly tolerant of poor input structures. Typical algorithm iteration times are longer than with nonlinear conjugate gradient or variable metric methods, but many fewer iterations are required for complete minimization. NEWTON is usually the best choice for minimizations to the 0.01 to 0.000001 kcal/mole/Ang level of RMS gradient convergence. Tests for directions of negative curvature can be removed, allowing NEWTON to be used for optimization to conformational transition state structures (this only works if the starting point is very close to the transition state). Input consists of a Tinker XYZ coordinates file; output is an updated set of minimized coordinates and minimization statistics.

NEWTROT

The NEWTROT program is similar to NEWTON except that it requires a .int file as input and then operates in terms of dihedral angles as the minimization variables. Since the dihedral space Hessian matrix of an arbitrary structure is often indefinite, this method will often not perform as well as the other, simpler dihedral angle based minimizers.

OPTIMIZE

The OPTIMIZE program performs a optimally conditioned variable metric minimization of an input structure over Cartesian coordinates using an algorithm due to William Davidon. The method does not perform line searches, but requires computation of energies and gradients as well as storage for an estimate of the inverse Hessian matrix. The program operates on Cartesian coordinates from a Tinker XYZ file. OPTIMIZE will typically converge somewhat faster and more completely than MINIMIZE. However, the need to store and manipulate a full inverse Hessian estimate limits its use to structures containing less than a few hundred atoms on workstation class machines. As with the other minimizers, OPTIMIZE needs input coordinates and an rms gradient cutoff criterion. The output coordinates are saved in updated .xyz files or as cycle files.

OPTIROT

The OPTIROT program is similar to OPTIMIZE except that it operates on dihedral angles starting from a Tinker INT internal coordinate file. This program is usually the preferred method for most dihedral angle optimization problems since Truncated Newton methods appear, in our hands, to lose some of their efficacy in moving from Cartesian to torsional coordinates.

OPTRIGID

The OPTRIGID program is similar to OPTIMIZE except that it operates on rigid bodies starting from a Tinker XYZ coordinate file and the rigid body atom group definitions found in the corresponding KEY file. Output is saved in an updated XYZ file or in cycle files.

PATH

A program that implements a variant of Elber's Lagrangian multiplier-based reaction path following algorithm. The program takes as input a pair of structural minima as Tinker XYZ files, and then generates a user specified number of points along a path through conformational space connecting the input structures. The intermediate structures are output as Tinker cycle files, and the higher energy intermediates can be used as input to a Newton-based optimization to locate conformational transition states.

PSS

Implements our version of a potential smoothing and search algorithm for the global optimization of molecular conformation. An initial structure in .xyz format is first minimized in Cartesian coordinates on a series of increasingly smoothed potential energy surfaces. Then the smoothing procedure is reversed with minimization on each successive surface starting from the coordinates of the minimum on the previous surface. A local search procedure is used during the backtracking to explore for alternative minima better than the one found during the current minimization. The final result is usually a very low energy conformation or, in favorable cases, the global energy minimum conformation. The minimum energy coordinate sets found on each surface during both the forward smoothing and backtracking procedures are placed in sequentially numbered cycle files.

PSSRIGID

This program implements the potential smoothing and search method as described above for the PSS program, but performs the computation in terms of keyfile-defined rigid body atom groups instead of Cartesian coordinates. Output is saved in numbered cycle files with the XYZ file format.

PSSROT

This program implements the potential smoothing and search method as described above for the PSS program, but performs the computation in terms of a set of user-specified dihedral angles instead of Cartesian coordinates. Output is saved in numbered cycle files with the INT file format.

SADDLE

A program for the location of a conformational transition state between two potential energy minima. SADDLE uses a conglomeration of ideas from the Bell-Crighton quadratic path and the Halgren-Lipscomb synchronous transit methods. The basic idea is to perform a nonlinear conjugate gradient optimization in a subspace orthogonal to a suitably defined reaction coordinate. The program requires as input the coordinates, as Tinker XYZ files, of the two minima and an rms gradient convergence criterion for the optimization. The current estimate of the transition state structure is written to the file TSTATE.XYZ. Crude transition state structures generated by SADDLE can sometimes be refined using the NEWTON program. Optionally, a scan of the interconversion pathway can be made at each major iteration.

SCAN

A program for general conformational search of an entire potential energy surface via a basin hopping method. The program takes as input a Tinker XYZ coordinates file which is then minimized to find the first local minimum for a search list. A series of activations along various normal modes from this initial minimum are used as seed points for additional minimizations. Whenever a previously unknown local minimum is located it is added to the search list. When all minima on the search list have been subjected to the normal mode activation without locating additional

new minima, the program terminates. The individual local minima are written to cycle files as they are discovered. While the SCAN program can be used on standard undeformed potential energy surfaces, we have found it to be most useful for quickly "scanning" a smoothed energy surface to enumerate the major basins of attraction spaning the entire surface.

SNIFFER

A program that implements the Sniffer global optimization algorithm of Butler and Slaminka, a discrete version of Griewank's global search trajectory method. The program takes an input Tinker XYZ coordinates file and shakes it vigorously via a modified dynamics trajectory before, hopefully, settling into a low lying minimum. Some trial and error is often required as the current implementation is sensitive to various parameters and tolerances that govern the computation. At present, these parameters are not user accessible, and must be altered in the source code. However, this method can do a good job of quickly optimizing conformation within a limited range of convergence.

TESTGRAD

The TESTGRAD program computes and compares the analytical and numerical first derivatives (i.e., the gradient vector) of the potential energy for a Cartesian coordinate input structure. The output can be used to test or debug the current potential or any added user defined energy terms.

TESTHESS

The TESTHESS program computes and compares the analytical and numerical second derivatives (i.e., the Hessian matrix) of the potential energy for a Cartesian coordinate input structure. The output can be used to test or debug the current potential or any added user defined energy terms.

TESTLIGHT

A program to compare the efficiency of different nonbonded neighbor methods for the current molecular system. The program times the computation of energy and gradient for the van der Waals and charge-charge electrostatic potential terms using a simple double loop over all interactions and using the Method of Lights algorithm to select neighbors. The results can be used to decide whether the Method of Lights has any CPU time advantage for the current structure. Both methods should give exactly the same answer in all cases, since the identical individual interactions are computed by both methods. The default double loop method is faster when cutoffs are not used, or when the cutoff sphere contains about half or more of the total system of unit cell. In cases where the cutoff sphere is much smaller than the system size, the Method of Lights can be much faster since it avoids unnecessary calculation of distances beyond the cutoff range.

TESTROT

The TESTROT program computes and compares the analytical and numerical first derivatives (i.e., the gradient vector) of the potential energy with respect to dihedral angles. Input is a Tinker INT internal coordinate file. The output can be used to test or debug the current potential functions or any added user defined energy terms.

TIMER

A simple program to provide timing statistics for energy function calls within the Tinker package. TIMER requires an input XYZ file and outputs the CPU time (or wall clock time, on some machine types) needed to perform a specified number of energy, gradient and Hessian evaluations.

TIMEROT

This program is similar to TIMER, only it operates over dihedral angles via input of a Tinker INT internal coordinate file. In the current version, the torsional Hessian is computed numerically from the analytical torsional gradient.

VIBRATE

A program to perform vibrational analysis by computing and diagonalizing the full Hessian matrix (i.e., the second partial derivatives) for an input structure (a Tinker XYZ file). Eigenvalues and eigenvectors of the mass weighted Hessian (i.e., the vibrational frequencies and normal modes) are also calculated. Structures corresponding to individual normal mode motions can be saved in cycle files.

VIBROT

The program VIBROT forms the torsional Hessian matrix via numerical differentiation of the analytical torsional gradient. The Hessian is then diagonalized and the eigenvalues are output. The present version does not compute the kinetic energy matrix elements needed to convert the Hessian into the torsional normal modes; this will be added in a later version. The required input is a Tinker INT internal coordinate file.

XTALFIT

The XTALFIT program is of use in the automated fitting of potential parameters to crystal structure and thermodynamic data. XTALFIT takes as input several crystal structures (Tinker XYZ files with unit cell parameters in corresponding KEY files) as well as information on lattice energies and dipole moments of monomers. The current version uses a nonlinear least squares optimization to fit van der Waals and electrostatic parameters to the input data. Bounds can be placed on the values of the optimization parameters.

XTALMIN

A program to perform full crystal minimizations. The program takes as input the structure coordinates and unit cell lattice parameters. It then alternates cycles of Newton-style optimization of the structure and conjugate gradient optimization of the crystal lattice parameters. This alternating minimization is slower than more direct optimization of all parameters at once, but is somewhat more robust in our hands. The symmetry of the original crystal is not enforced, so interconversion of crystal forms may be observed in some cases.

CHAPTER

FIVE

ANALYSIS & UTILITY PROGRAMS & SCRIPTS

This section of the manual contains a brief description of each of the Tinker structure manipulation, geometric calculation and auxiliary programs. A detailed example showing how to run each program is included in a later section. The programs listed below are all part of the main, supported distribution. Additional source code for various unsupported programs can be found in the /other directory of the Tinker distribution.

ARCHIVE

A program for concatenating Tinker cycle files into a single archive file; useful for storing the intermediate results of minimizations, dynamics trajectories, and so on. The program can also extract individual cycle files from a Tinker archive.

CORRELATE

A program to compute time correlation functions from collections of Tinker cycle files. Its use requires a user supplied function property that computes the value of the property for which a time correlation is desired for two input structures. A sample routine is supplied that computes either a velocity autocorrelation function or an rms structural superposition as a function of time. The main body of the program organizes the overall computation in an efficient manner and outputs the final time correlation function.

CRYSTAL

A program for the manipulation of crystal structures including interconversion of fractional and Cartesian coordinates, generation of the unit cell from an asymmetric unit, and building of a crystalline block of specified size via replication of a single unit cell. The present version can handle about 25 of the most common space groups, others can easily be added as needed by modification of the routine symmetry.

DIFFUSE

A program to compute the self-diffusion constant for a homogeneous liquid via the Einstein equation. A previously saved dynamics trajectory is read in and ``unfolded" to reverse translation of molecules due to use of periodic boundary conditions. The average motion over all molecules is then used to compute the self-diffusion constant. While the current program assumes a homogeneous system, it should be easy to modify the code to handle diffusion of individual molecules or other desired effects.

DISTGEOM

A program to perform distance geometry calculations using variations on the classic metric matrix method. A user specified number of structures consistent with keyfile input distance and dihedral restraints is generated. Bond length and angle restraints are derived from the input structure. Trial distances between the triangle smoothed lower and upper bounds can be chosen via any of several metrization methods, including a very effective partial random pairwise scheme. The correct radius of gyration of the structure is automatically maintained by choosing trial distances from Gaussian distributions of appropriate mean and width. The initial embedded structures can be further refined against a geometric restraint-only potential using either a sequential minimization protocol or simulated annealing.

DOCUMENT

The DOCUMENT program is provided as a minimal listing and documentation tool. It operates on the Tinker source code, either individual files or the complete source listing produced by the command script listing.make, to generate lists of routines, common blocks or valid keywords. In addition, the program has the ability to output a formatted parameter listing from the standard Tinker parameter files.

INTEDIT

A program to allow interactive inspection and alteration of the internal coordinate definitions and values of a Tinker structure. If the structure is altered, the user has the option to write out a new internal coordinates file upon exit.

INTXYZ

A program to convert a Tinker .int internal coordinates formatted file into a Tinker .xyz Cartesian coordinates formatted file.

MOL2XYZ

A program for converting a Tripos Sybyl MOL2 file into a Tinker XYZ Cartesian coordinate file. The current version of the program does not attempt to convert the Sybyl atoms types into the active Tinker force field types, i.e., all atoms types are simply set to zero.

NUCLEIC

A program for automated building of nucleic acid structures. Upon interactive input of a nucleotide sequence with optional phosphate backbone angles, the program builds internal and Cartesian coordinates. Standard bond lengths and angles are used. Both DNA and RNA sequences are supported as are A-, B- and Z-form structures. Double helixes of complementary sequence can be automatically constructed via a rigid docking of individual strands.

PDBXYZ

A program for converting a Brookhaven Protein Data Bank file (a PDB file) into a Tinker .xyz Cartesian coordinate file. If the PDB file contains only protein/peptide amino acid residues, then standard protein connectivity is assumed, and transferred to the .xyz file. For non-protein portions of the PDB file, atom connectivity is determined by the program based on interatomic distances. The program also has the ability to add or remove hydrogen atoms from a protein as required by the force field specified during the computation.

POLARIZE

A program for computing molecular polarizability from an atom-based distributed model of polarizability. A damped interaction model due to Thole is optionally via keyfile settings. A Tinker .xyz file is required as input. The output consists of the overall polarizability tensor in the global coordinates and its eigenvalues.

PRMEDIT

A program for formatting and renumbering Tinker force field parameter files. When atom types or classes are added to a parameter file, this utility program has the ability to renumber all the atom records sequentially, and alter type and class numbers in all other parameter entries to maintain consistency.

PROTEIN

A program for automated building of peptide and protein structures. Upon interactive input of an amino acid sequence with optional phi/psi/omega/chi angles, D/L chirality, etc., the program builds internal and Cartesian coordinates. Standard bond lengths and angles are assumed for the peptide. The program will optionally convert the structure to a cyclic peptide, or add either or both N- and C-terminal capping groups. Atom type numbers are automatically assigned for the specified force field. The final coordinates and a sequence file are produced as the output.

RADIAL

A program to compute the pair radial distribution function between two atom types. The user supplies the two atom names for which the distribution function is to be computed, and the width of the distance bins for data analysis. A previously saved dynamics trajectory is read as input. The raw radial distribution and a spline smoothed version are then output from zero to a distance equal to half the minimum periodic box dimension. The atom names are matched to the atom name column of the Tinker .xyz file, independent of atom type.

SPACEFILL

A program to compute the volume and surface areas of molecules. Using a modified version of Connolly's original analytical description of the molecular surface, the program determines either the van der Waals, accessible or molecular (contact/reentrant) volume and surface area. Both surface area and volume are broken down into their geometric components, and surface area is decomposed into the convex contribution for each individual atom. The probe radius is input as a user option, and atomic radii can be set via the keyword file. If Tinker archive files are used as input, the program will compute the volume and surface area of each structure in the input file.

SPECTRUM

A program to compute a power spectrum from velocity autocorrelation data. As input, this program requires a velocity autocorrelation function as produced by the CORRELATE program. This data, along with a user input time step, are Fourier transformed to generate the spectral intensities over a wavelength range. The result is a power spectrum, and the positions of the bands are those predicted for an infrared or Raman spectrum. However, the data is not weighted by molecular dipole moment derivatives as would be required to produce correct IR intensities.

SUPERPOSE

A program to superimpose two molecular structures in 3-dimensions. A variety of options for input of the atom sets to be used during the superposition are presented interactively to the user. The superposition can be mass-weighted if desired, and the coordinates of the second structure

superimposed on the first structure are optionally output. If Tinker archive files are used as input, the program will compute all pairwise superpositions between structures in the input files.

XYZEDIT

A program that performs and of a variety of manipulations on an input Tinker .xyz Cartesian coordinates formatted file. The present version of the program has the following interactively selectable options: (1) Offset the Numbers of the Current Atoms, (2) Deletion of Individual Specified Atoms, (3) Deletion of Specified Types of Atoms, (4) Deletion of Atoms outside Cutoff Range, (5) Insertion of Individual Specified Atoms, (6) Replace Old Atom Type with a New Type, (7) Assign Connectivities based on Distance, (8) Convert Units from Bohrs to Angstroms, (9) Invert thru Origin to give Mirror Image, (10) Translate Center of Mass to the Origin, (11) Translate a Specified Atom to the Origin, (12) Translate and Rotate to Inertial Frame, (13) Move to Specified Rigid Body Coordinates, (14) Create and Fill a Periodic Boundary Box, (15) Soak Current Molecule in Box of Solvent, (16) Append another XYZ file to Current One. In most cases, multiply options can be applied sequentially to an input file. At the end of the editing process, a new version of the original .xyz file is written as output.

XYZINT

A program for converting a Tinker .xyz Cartesian coordinate formatted file into a Tinker .int internal coordinates formatted file. This program can optionally use an existing internal coordinates file as a template for the connectivity information.

XYZMOL2

A program to convert a Tinker .xyz Cartesian coordinates file into a Tripos Sybyl MOL2 file. The conversion generates only the MOLECULE, ATOM, BOND and SUBSTRUCTURE record type in the MOL2 file. Generic Sybyl atom types are used in most cases; while these atom types may need to be altered in some cases, Sybyl is usually able to correctly display the resulting MOL2 file.

XYZPDB

A program for converting a Tinker .xyz Cartesian coordinate file into a Brookhaven Protein Data Bank file (a PDB file).

CHAPTER

SIX

FORCE FIELD PARAMETER SETS

The Tinker package is distributed with several force field parameter sets, implementing a selection of widely used literature force fields as well as the Tinker force field currently under construction in the Ponder lab. We try to exactly reproduce the intent of the original authors of our distributed, third-party force fields. In all cases the parameter sets have been validated against literature reports, results provided by the original developers, or calculations made with the authentic programs. With the few exceptions noted below, Tinker calculations can be treated as authentic results from the genuine force fields. A brief description of each parameter set, including some still in preparation and not distributed with the current version, is provided below with lead literature references for the force field:

AMOEBA.PRM

Parameters for the AMOEBA polarizable atomic multipole force field. As of the current Tinker release, we have completed parametrization for a number of ions and small organic molecules. For further information, or if you are interested in developing or testing parameters for other small molecules, please contact the Ponder lab.

P. Ren and J. W. Ponder, A Consistent Treatment of Inter- and Intramolecular Polarization in Molecular Mechanics Calculations, J. Comput. Chem., 23, 1497-1506 (2002)

P. Ren and J. W. Ponder, Polarizable Atomic Multipole Water Model for Molecular Mechanics Simulation, J. Phys. Chem. B, 107, 5933-5947 (2003)

P. Ren and J. W. Ponder, Ion Solvation Thermodynamics from Simulation with a Polarizable Force Field, A. Grossfield, J. Am. Chem. Soc., 125, 15671-15682 (2003)

AMOEBAPRO.PRM

Preliminary protein parameters for the AMOEBA polarizable atomic multipole force field. While the distributed parameters are still subject to minor alteration as we continue validation, they are now stable enough for other groups to begin using them. For further information, or if you are interested in testing the protein parameter set, please contact the Ponder lab.

J. W. Ponder and D. A. Case, Force Fields for Protein Simulation, Adv. Prot. Chem., 66, 27-85 (2003)

P. Ren and J. W. Ponder, Polarizable Atomic Multipole-based Potential for Proteins: Model and Parameterization, in preparation

AMBER94.PRM

AMBER ff94 parameters for proteins and nucleic acids. Note that with their "Cornell" force field, the Kollman group has devised separate, fully independent partial charge values for each of the N- and C-terminal amino acid residues. At present, the terminal residue charges for Tinker's version maintain the correct formal charge, but redistributed somewhat at the alpha carbon atoms from the original Kollman group values. The total magnitude of the redistribution is less than 0.01 electrons in most cases.

W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, Jr., D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell and P. A. Kollman, A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules, J. Am. Chem. Soc., 117, 5179-5197 (1995) [ff94]

G. Moyna, H. J. Williams, R. J. Nachman and A. I. Scott, Conformation in Solution and Dynamics of a Structurally Constrained Linear Insect Kinin Pentapeptide Analogue, Biopolymers, 49, 403-413 (1999) [AIB charges]

W. S. Ross and C. C. Hardin, Ion-Induced Stabilization of the G-DNA Quadruplex: Free Energy Perturbation Studies, J. Am. Chem. Soc., 116, 4363-4366 (1994) [alkali metal ions]

J. Aqvist, Ion-Water Interaction Potentials Derived from Free Energy Perturbation Simulations, J. Phys. Chem., 94, 8021-8024, 1990 [alkaline earth Ions, radii adapted for Amber combining rule]

Current force field parameter values and suggested procedures for development of parameters for additional molecules are available from the Amber web site in the Case lab at Scripps, http://amber.scripps.edu/

AMBER96.PRM

AMBER ff96 parameters for proteins and nucleic acids. The only change from the ff94 parameter set is in the torsional parameters for the protein phi/psi angles. These values were altered to give better agreement with changes of ff96 with LMP2 QM results from the Friesner lab on alanine dipeptide and tetrapeptide.

P. Kollman, R. Dixon, W. Cornell, T. Fox, C. Chipot and A. Pohorille, The Development/ Application of a 'Minimalist' Organic/Biochemical Molecular Mechanic Force Field using a Combination of ab Initio Calculations and Experimental Data, in Computer Simulation of Biomolecular Systems, W. F. van Gunsteren, P. K. Weiner, A. J. Wilkinson, eds., Volume 3, 83-96 (1997) [ff96]

Current force field parameter values and suggested procedures for development of parameters for additional molecules are available from the Amber web site in the Case lab at Scripps, http://amber.scripps.edu/

AMBER98.PRM

AMBER ff98 parameters for proteins and nucleic acids. The only change from the ff94 parameter set is in the glycosidic torsional parameters that control sugar pucker.

T. E. Cheatham III, P. Cieplak and P. A. Kollman, A Modified Version of the Cornell et al. Force Field with Improved Sugar Pucker Phases and Helical Repeat, J. Biomol. Struct. Dyn., 16, 845-862 (1999)

Current force field parameter values and suggested procedures for development of parameters for additional molecules are available from the Amber web site in the Case lab at Scripps, http://amber.scripps.edu/

AMBER99.PRM

AMBER ff99 parameters for proteins and nucleic acids. The original partial charges from the ff94 parameter set are retained, but many of the bond, angle and torsional parameters have been revised to provide better general agreement with experiment.

J. Wang, P. Cieplak and P. A. Kollman, How Well Does a Restrained Electrostatic Potential (RESP) Model Perform in Calcluating Conformational Energies of Organic and Biological Molecules?, J. Comput. Chem., 21, 1049-1074 (2000)

Current force field parameter values and suggested procedures for development of parameters for additional molecules are available from the Amber web site in the Case lab at Scripps, http://amber.scripps.edu/

CHARMM19.PRM

CHARMM19 united-atom parameters for proteins. The nucleic acid parameter are not yet implemented. There are some differences between authentic CHARMM19 and the Tinker version due to replacement of CHARMM impropers by torsions for cases that involve atoms not bonded to the trigonal atom and Tinker's use of all possible torsions across a bond instead of a single torsion per bond.

E. Neria, S. Fischer and M. Karplus, Simulation of Activation Free Energies in Molecular Systems, J. Chem. Phys., 105, 1902-1921 (1996)

L. Nilsson and M. Karplus, Empirical Energy Functions for Energy Minimizations and Dynamics of Nucleic Acids, J. Comput. Chem., 7, 591-616 (1986)

W. E. Reiher III, Theoretical Studies of Hydrogen Bonding, Ph.D. Thesis, Department of Chemistry, Harvard University, Cambridge, MA, 1985

CHARMM22.PRM

CHARMM22 all-atom parameters for proteins and lipids. Most of the nucleic acid and small model compound parameters are not yet implemented. We plan to provide these additional parameters in due course.

N. Foloppe and A. D. MacKerell, Jr., All-Atom Empirical Force Field for Nucleic Acids: 1) Parameter Optimization Based on Small Molecule and Condensed Phase Macromolecular Target Data, J. Comput. Chem., 21, 86-104 (2000) [CHARMM27]

N. Banavali and A. D. MacKerell, Jr., All-Atom Empirical Force Field for Nucleic Acids: 2) Application to Molecular Dynamics Simulations of DNA and RNA in Solution, J. Comput. Chem., 21, 105-120 (2000)

A. D. MacKerrell, Jr., et al., All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins, J. Phys. Chem. B, 102, 3586-3616 (1998) [CHARMM22]

A. D. MacKerell, Jr., J. Wiorkeiwicz-Kuczera and M. Karplus, An All-Atom Empirical Energy Function for the Simulation of Nucleic Acids, J. Am. Chem. Soc., 117, 11946-11975 (1995)

S. E. Feller, D. Yin, R. W. Pastor and A. D. MacKerell, Jr., Molecular Dynamics Simulation of Unsaturated Lipids at Low Hydration: Parametrization and Comparison with Diffraction Studies, Biophysical Journal, 73, 2269-2279 (1997) [alkenes]

R. H. Stote and M. Karplus, Zinc Binding in Proteins and Solution - A Simple but Accurate Nonbonded Representation, Proteins, 23, 12-31 (1995) [zinc ion]

Current and legacy parameter values are available from the CHARMM force field web site on Alex MacKerell's Research Interests page at the University of Maryland School of Pharmacy, https://rxsecure.umaryland.edu/research/amackere/research.html/

DUDEK.PRM

Protein-only parameters for the early 1990's Tinker force field with multipole values of Dudek and Ponder. The current file contains only the multipole values from the 1995 paper by Dudek and Ponder. This set is now superceeded by the more recent Tinker force field developed by Pengyu Ren (see WATER.PRM, below).

M. J. Dudek and J. W. Ponder, Accurate Electrostatic Modelling of the Intramolecular Energy of Proteins, J. Comput. Chem., 16, 791-816 (1995)

ENCAD.PRM

ENCAD parameters for proteins and nucleic acids. (in preparation)

M. Levitt, M. Hirshberg, R. Sharon and V. Daggett, Potential Energy Function and Parameters for Simulations of the Molecular Dynamics of Protein and Nucleic Acids in Solution, Comp. Phys. Commun., 91, 215-231 (1995)

M. Levitt, M. Hirshberg, R. Sharon, K. E. Laidig and V. Daggett, Calibration and Testing of a Water Model for Simulation of the Molecular Dynamics of Protein and Nucleic Acids in Solution, J. Phys. Chem. B, 101, 5051-5061 (1997) [F3C water]

HOCH.PRM

Simple NMR-NOE force field of Hoch and Stern.

J. C. Hoch and A. S. Stern, A Method for Determining Overall Protein Fold from NMR Distance Restraints, J. Biomol. NMR, 2, 535-543 (1992)

MM2.PRM

Full MM2(1991) parameters including ?-systems. The anomeric and electronegativity correction terms included in some later versions of MM2 are not implemented.

N. L. Allinger, Conformational Analysis. 130. MM2. A Hydrocarbon Force Field Utilizing V1 and V2 Torsional Terms, J. Am. Chem. Soc., 99, 8127-8134 (1977)

J. T. Sprague, J. C. Tai, Y. Yuh and N. L. Allinger, The MMP2 Calculational Method, J. Comput. Chem., 8, 581-603 (1987)

J. C. Tai and N. L. Allinger, Molecular Mechanics Calculations on Conjugated Nitrogen-Containing Heterocycles, J. Am. Chem. Soc., 110, 2050-2055 (1988)

J. C. Tai, J.-H. Lii and N. L. Allinger, A Molecular Mechanics (MM2) Study of Furan, Thiophene, and Related Compounds, J. Comput. Chem., 10, 635-647 (1989)

N. L. Allinger, R. A. Kok and M. R. Imam, Hydrogen Bonding in MM2, J. Comput. Chem., 9, 591-595 (1988)

L. Norskov-Lauritsen and N. L. Allinger, A Molecular Mechanics Treatment of the Anomeric Effect, J. Comput. Chem., 5, 326-335 (1984)

All parameters distributed with Tinker are from the "MM2 (1991) Parameter Set", as provided by N. L. Allinger, University of Georgia

MM3.PRM

Full MM3(2000) parameters including pi-systems. The directional hydrogen bonding term and electronegativity bond length corrections are implemented, but the anomeric and Bohlmann correction terms are not implemented.

N. L. Allinger, Y. H. Yuh and J.-H. Lii, Molecular Mechanics. The MM3 Force Field for Hydrocarbons. 1, J. Am. Chem. Soc., 111, 8551-8566 (1989)

J.-H. Lii and N. L. Allinger, Molecular Mechanics. The MM3 Force Field for Hydrocarbons. 2. Vibrational Frequencies and Thermodynamics, J. Am. Chem. Soc., 111, 8566-8575 (1989)

J.-H. Lii and N. L. Allinger, Molecular Mechanics. The MM3 Force Field for Hydrocarbons. 3. The van der Waals' Potentials and Crystal Data for Aliphatic and Aromatic Hydrocarbons, J. Am. Chem. Soc., 111, 8576-8582 (1989)

N. L. Allinger, H. J. Geise, W. Pyckhout, L. A. Paquette and J. C. Gallucci, Structures of Norbornane and Dodecahedrane by Molecular Mechanics Calculations (MM3), X-ray Crystallography, and Electron Diffraction, J. Am. Chem. Soc., 111, 1106-1114 (1989) [stretch-torsion cross term]

N. L. Allinger, F. Li and L. Yan, Molecular Mechanics. The MM3 Force Field for Alkenes, J. Comput. Chem., 11, 848-867 (1990)

N. L. Allinger, F. Li, L. Yan and J. C. Tai, Molecular Mechanics (MM3) Calculations on Conjugated Hydrocarbons, J. Comput. Chem., 11, 868-895 (1990)

J.-H. Lii and N. L. Allinger, Directional Hydrogen Bonding in the MM3 Force Field. I, J. Phys. Org. Chem., 7, 591-609 (1994)

J.-H. Lii and N. L. Allinger, Directional Hydrogen Bonding in the MM3 Force Field. II, J. Comput. Chem., 19, 1001-1016 (1998)

All parameters distributed with Tinker are from the "MM3 (2000) Parameter Set", as provided by N. L. Allinger, University of Georgia, August 2000

MM3PRO.PRM

Protein-only version of the MM3 parameters.

J.-H. Lii and N. L. Allinger, The MM3 Force Field for Amides, Polypeptides and Proteins, J. Comput. Chem., 12, 186-199 (1991)

OPLSUA.PRM

Complete OPLS-UA with united-atom parameters for proteins and many classes of organic molecules. Explicit hydrogens on polar atoms and aromatic carbons.

W. L. Jorgensen and J. Tirado-Rives, The OPLS Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin, J. Am. Chem. Soc., 110, 1657-1666 (1988) [peptide and proteins]

W. L. Jorgensen and D. L. Severance, Aromatic-Aromatic Interactions: Free Energy Profiles for the Benzene Dimer in Water, Chloroform, and Liquid Benzene, J. Am. Chem. Soc., 112, 4768-4774 (1990) [aromatic hydrogens]

S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta, Jr. and P. Weiner, A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins, J. Am. Chem. Soc., 106, 765-784 (1984) [united-atom "AMBER/OPLS" local geometry]

S. J. Weiner, P. A. Kollman, D. T. Nguyen and D. A. Case, An All Atom Force Field for Simulations of Proteins and Nucleic Acids, J. Comput. Chem., 7, 230-252 (1986) [all-atom "AMBER/OPLS" local geometry]

L. X. Dang and B. M. Pettitt, Simple Intramolecular Model Potentials for Water, J. Phys. Chem., 91, 3349-3354 (1987) [flexible TIP3P and SPC water]

W. L. Jorgensen, J. D. Madura and C. J. Swenson, Optimized Intermolecular Potential Functions for Liquid Hydrocarbons, J. Am. Chem. Soc., 106, 6638-6646 (1984) [hydrocarbons]

W. L. Jorgensen, E. R. Laird, T. B. Nguyen and J. Tirado-Rives, Monte Carlo Simulations of Pure Liquid Substituted Benzenes with OPLS Potential Functions, J. Comput. Chem., 14, 206-215 (1993) [substituted benzenes]

E. M. Duffy, P. J. Kowalczyk and W. L. Jorgensen, Do Denaturants Interact with Aromatic Hydrocarbons in Water?, J. Am. Chem. Soc., 115, 9271-9275 (1993) [benzene, naphthalene, urea, guanidinium, tetramethyl ammonium]

W. L. Jorgensen and C. J. Swenson, Optimized Intermolecular Potential Functions for Amides and Peptides. Structure and Properties of Liquid Amides, J. Am. Chem. Soc., 106, 765-784 (1984) [amides]

W. L. Jorgensen, J. M. Briggs and M. L. Contreras, Relative Partition Coefficients for Organic Solutes form Fluid Simulations, J. Phys. Chem., 94, 1683-1686 (1990) [chloroform, pyridine, pyrazine, pyrimidine]

J. M. Briggs, T. B. Nguyen and W. L. Jorgensen, Monte Carlo Simulations of Liquid Acetic Acid and Methyl Acetate with the OPLS Potential Functions, J. Phys. Chem., 95, 3315-3322 (1991) [acetic acid, methyl acetate]

H. Liu, F. Muller-Plathe and W. F. van Gunsteren, A Force Field for Liquid Dimethyl Sulfoxide and Physical Properties of Liquid Dimethyl Sulfoxide Calculated Using Molecular Dynamics Simulation, J. Am. Chem. Soc., 117, 4363-4366 (1995) [dimethyl sulfoxide]

J. Gao, X. Xia and T. F. George, Importance of Bimolecular Interactions in Developing Empirical Potential Functions for Liquid Ammonia, J. Phys. Chem., 97, 9241-9246 (1993) [ammonia]

J. Aqvist, Ion-Water Interaction Potentials Derived from Free Energy Perturbation Simulations, J. Phys. Chem., 94, 8021-8024 (1990) [metal ions]

W. S. Ross and C. C. Hardin, Ion-Induced Stabilization of the G-DNA Quadruplex: Free Energy Perturbation Studies, J. Am. Chem. Soc., 116, 4363-4366 (1994) [alkali metal ions]

J. Chandrasekhar, D. C. Spellmeyer and W. L. Jorgensen, Energy Component Analysis for Dilute Aqueous Solutions of Li+, Na+, F-, and Cl- Ions, J. Am. Chem. Soc., 106, 903-910 (1984) [halide ions]

Most parameters distributed with Tinker are from "OPLS and OPLS-AA Parameters for Organic Molecules, Ions, and Nucleic Acids" as provided by W. L. Jorgensen, Yale University, October 1997

OPLSAA.PRM

OPLS-AA force field with all-atom parameters for proteins and many general classes of organic molecules.

W. L. Jorgensen, D. S. Maxwell and J. Tirado-Rives, Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids, J. Am. Chem. Soc., 117, 11225-11236 (1996)

D. S. Maxwell, J. Tirado-Rives and W. L. Jorgensen, A Comprehensive Study of the Rotational Energy Profiles of Organic Systems by Ab Initio MO Theory, Forming a Basis for Peptide Torsional Parameters, J. Comput. Chem., 16, 984-1010 (1995)

W. L. Jorgensen and N. A. McDonald, Development of an All-Atom Force Field for Heterocycles. Properties of Liquid Pyridine and Diazenes, THEOCHEM-J. Mol. Struct., 424, 145-155 (1998)

N. A. McDonald and W. L. Jorgensen, Development of an All-Atom Force Field for Heterocycles. Properties of Liquid Pyrrole, Furan, Diazoles, and Oxazoles, J. Phys. Chem. B, 102, 8049-8059 (1998)

R. C. Rizzo and W. L. Jorgensen, OPLS All-Atom Model for Amines: Resolution of the Amine Hydration Problem, J. Am. Chem. Soc., 121, 4827-4836 (1999)

M. L. P. Price, D. Ostrovsky and W. L. Jorgensen, Gas-Phase and Liquid-State Properties of Esters, Nitriles, and Nitro Compounds with the OPLS-AA Force Field, J. Comput. Chem., 22, 1340-1352 (2001)

All parameters distributed with Tinker are from "OPLS and OPLS-AA Parameters for Organic Molecules, Ions, and Nucleic Acids" as provided by W. L. Jorgensen, Yale University, October 1997

OPLSAAL.PRM

An improved OPLS-AA parameter set for proteins in which the only change is a reworking of many of the backbone and sidechain torsional parameters to give better agreement with LMP2 QM calculations. This parameter set is also known as OPLS(2000).

G. A. Kaminsky, R. A. Friesner, J. Tirado-Rives and W. L. Jorgensen, Evaluation and Reparametrization of the OPLS-AA Force Field for Proteins via Comparison with Accurate Quantum Chemical Calculations on Peptides, J. Phys. Chem. B, 105, 6474-6487 (2001)

SMOOTH.PRM

Version of OPLS-UA for use with potential smoothing. Largely adapted largely from standard OPLS-UA parameters with modifications to the vdw and improper torsion terms.

R. V. Pappu, R. K. Hart and J. W. Ponder, Analysis and Application of Potential Energy Smoothing and Search Methods for Global Optimization, J. Phys, Chem. B, 102, 9725-9742 (1998) [smoothing modifications]

SMOOTHAA.PRM

Version of OPLS-AA for use with potential smoothing. Largely adapted largely from standard OPLS-AA parameters with modifications to the vdw and improper torsion terms.

R. V. Pappu, R. K. Hart and J. W. Ponder, Analysis and Application of Potential Energy Smoothing and Search Methods for Global Optimization, J. Phys, Chem. B, 102, 9725-9742 (1998) [smoothing modifications]

WATER.PRM

The AMOEBA water parameters for a polarizable atomic multipole electrostatics model. This model is equal or better to the best available water models for many bulk and cluster properties.

P. Ren and J. W. Ponder, A Polarizable Atomic Multipole Water Model for Molecular Mechanics Simulation, J. Phys. Chem. B, 107, 5933-5947 (2003)

P. Ren and J. W. Ponder, Ion Solvation Thermodynamics from Simulation with a Polarizable Force Field, A. Grossfield, J. Am. Chem. Soc., 125, 15671-15682 (2003)

P. Ren and J. W. Ponder, Temperature and Pressure Dependence of the AMOEBA Water Model, J. Phys. Chem. B, 108, 13427-13437 (2004)

An earlier version the AMOEBA water model is described in: Yong Kong, Multipole Electrostatic Methods for Protein Modeling with Reaction Field Treatment, Biochemistry & Molecular Biophysics, Washington University, St. Louis, August, 1997 [available from http://dasher.wustl.edu/ponder/]

CHAPTER

SEVEN

SPECIAL FEATURES & METHODS

This section contains several short notes with further information about Tinker methodology, algorithms and special features. The discussion is not intended to be exhaustive, but rather to explain features and capabilities so that users can make more complete use of the package.

7.1 File Version Numbers

All of the input and output file types routinely used by the Tinker package are capable of existing as multiple versions of a base file name. For example, if the program XYZINT is run on the input file molecule.xyz, the output internal coordinates file will be written to molecule.int. If a file named molecule.int is already present prior to running XYZINT, then the output will be written instead to the next available version, in this case to molecule.int_2. In fact the output is generally written to the lowest available, previously unused version number (molecule.int_3, molecule.int_4, etc., as high as needed). Input file names are handled similarly. If simply molecule or molecule.xyz is entered as the input file name upon running XYZINT, then the highest version of molecule.xyz will be used as the actual input file. If an explicit version number is entered as part of the input file name, then the specified version will be used as the input file.

The version number scheme will be recognized by many older users as a holdover from the VMS origins of the first version of the Tinker software. It has been maintained to make it easier to chain together multiple calculations that may create several new versions of a given file, and to make it more difficult to accidently overwrite a needed result. The version scheme applies to most uses of many common Tinker file types such as .xyz, .int, .key, .arc. It is not used when an overwritten file update is obviously the correct action, for example, the .dyn molecular dynamics restart files. For those users who prefer a more Unix-like operation, and do not desire use of file versions, this feature can be turned off by adding the NOVERSION keyword to the applicable Tinker keyfile.

The version scheme as implemented in Tinker does have two known quirks. First, it becomes impossible to directly use the original unversioned copy of a file if higher version numbers are present. For example, if the files molecule.xyz and molecule.xyz_2 both exist, then molecule.xyz cannot be accessed as input by XYZINT. If molecule.xyz is entered in response to the input file name question, molecule.xyz_2 (or the highest present version number) will be used as input. The only workaround is to copy or rename molecule.xyz to something else, say molecule.new, and use that name for the input file. Secondly, missing version numbers always end the search for the highest available version number; i.e., version numbers are assumed to be consecutive and without gaps. For example, if molecule.xyz, molecule.xyz_2 and molecule.xyz_4 are present, but

not molecule.xyz_3, then molecule.xyz_2 will be used as input to XYZINT if molecule is given as the input file name. Similarly, output files will fill in gaps in an already existing set of file versions.

7.2 Command Line Options

Many operating systems or compiler supplied-libraries make available something like the standard Unix iargc and getarg routines for capturing command line arguments. On these machines most of the Tinker programs support a selection of command line arguments and options. The name of the keyfile to be used for a calculation is read from the argument following a -k (equivalent to either -key or -keyfile, case insensitive) command line argument. Note that the -k options can appear anywhere on the command line following the executable name. All other command line arguments, excepting the name of the executable program itself, are treated as input arguments. These input arguments are read from left to right and interpreted in order as the answers to questions that would be asked by an interactive invocation of the same Tinker program. For example, the following command line:

newton molecule -k test a a 0.01

will invoke the NEWTON program on the structure file molecule.xyz using the keyfile test.key, automatic mode [a] for both the method and preconditioning, and 0.01 for the RMS gradient per atom termination criterion in kcal/mole/Ang. Provided that the force field parameter set, etc. is provided in test.key, the above compuation will procede directly from the command line invocation without further interactive input.

7.3 Use on Windows Systems

Tinker executables for Microsoft PC systems should be run from the DOS or Command Prompt window available under the various versions of Windows. The Tinker executable directory should be added to your path via the autoexec.bat file or similar. If a Command Prompt window, set the number of scrollable lines to a very large number, so that you will be able to inspect screen output after it moves by. Alternatively, Tinker programs which generate large amounts of screen output should be run such that output will be redirected to a file. This can be accomplished by running the Tinker program in batch mode or by using the build-in Unix-like output redirection. For example, the command:

dynamic < molecule.inp > molecule.log

will run the Tinker dynamic program taking input from the file molecule.inp and sending output to molecule.log. Also note that command line options as described above are available with the distributed Tinker executables.

If the distributed Tinker executables are run directly from Windows by double clicking on the program icon, then the program will run in its own window. However, upon completion of the program the window will close and screen output will be lost. Any output files written by the program will, of course, still be available. The Windows behavior can be changed by adding the EXIT-PAUSE keyword to the keyfile. This keyword causes the executation window to remain open after completion until the “Return/Enter” key is pressed.

An alternative to Command Prompt windows is to use the PowerShell window available on Windows 10 systems, which provides a better emulation of many of the standard features of Linux shells and MacOS Terminal.

Yet another alternative, particularly attractive to those already familiar with Linux or Unix systems, is to download the Cygwin package currently available under GPL license from the site http://source.redhat.com/cygwin/. The cygwin tools provide many of the GNU tools, including a bash shell window from which Tinker programs can be run.

Finally on Windows 10 systems, it is possible to download and install the Windows Subsystem for Linux (WSL), and then run the Tinker Linux executables from within WSL.

7.4 Use on MacOS Systems

The command line versions of the Tinker executables are best run on MacOS in a "Terminal" application window where behavior is essentially identical to that in a Linux terminal.

7.5 Atom Types vs. Atom Classes

Manipulation of atom types and the proliferation of parameters as atoms are further subdivided into new types is the bane of force field calculation. For example, if each topologically distinct atom arising from the 20 natural amino acids is given a different atom type, then about 300 separate type are required (this ignores the different N- and C-terminal forms of the residues, diastereotopic hydrogens, etc.). However, all these types lead to literally thousands of different force field parameters. In fact, there are many thousands of distinct torsional parameters alone. It is impossible at present to fully optimize each of these parameters; and even if we could, a great many of the parameters would be nearly identical. Two somewhat complimentary solutions are available to handle the proliferation of parameters. The first is to specify the molecular fragments to which a given parameter can be applied in terms of a chemical structure language, SMILES strings for example.

A second general approach is to use hierarchical cascades of parameter groups. Tinker uses a simple version of this scheme. Each Tinker force field atom has both an atom type number and an atom class number. The types are subsets of the atom classes, i.e., several different atom types can belong to the same atom class. Force field parameters that are somewhat less sensitive to local environment, such as local geometry terms, are then provided and assigned based on atom class. Other energy parameters, such as electrostatic parameters, that are very environment dependent are assigned over the atom types. This greatly reduces the number of independent multiple-atom parameters like the four-atom torsional parameters.

7.6 Calculations on Partial Structures

Two methods are available for performing energetic calculations on portions or substructures within a full molecular system. Tinker allows division of the entire system into active and inactive parts which can be defined via keywords. In subsequent calculations, such as minimization or dynamics, only the active portions of the system are allowed to move. The force field engine responds to the active/inactive division by computing all energetic interactions involving at least one active atom; i.e., any interaction whose energy can change with the motion of one or more active atoms is computed.

The second method for partial structure computation involves dividing the original system into a set of atom groups. As before, the groups can be specified via appropriate keywords. The current Tinker implementation allows specification of up to a maximum number of groups as given in the sizes.i dimensioning file. The groups must be disjoint in that no atom can belong to more than one group. Further keywords allow the user to specify which intra- and intergroup sets of energetic interactions will contribute to the total force field energy. Weights for each set of interactions in the total energy can also be input. A specific energetic interaction is assigned to a particular intra- or intergroup set if all the atoms involved in the interaction belong to the group (intra-) or pair of groups (inter-). Interactions involving atoms from more than two groups are not computed.

Note that the groups method and active/inactive method use different assignment procedures for individual interactions. The active/inactive scheme is intended for situations where only a portion of a system is allowed to move, but the total energy needs to reflect the presence of the remaining inactive portion of the structure. The groups method is intended for use in rigid body calculations, and is needed for certain kinds of free energy perturbation calculations.

7.7 Metal Complexes and Hypervalent Species

The distribution version of Tinker comes dimensioned for a maximum atomic coordination number of four as needed for standard organic compounds. In order to use Tinker for calculations on species containing higher coordination numbers, simply change the value of the parameter maxval in the master dimensioning file sizes.i and rebuilt the package. Note that this parameter value should not be set larger than necessary since large values can slow the execution of portions of some Tinker programs.

Many molecular mechanics approaches to inorganic and metal structures use an angle bending term which is softer than the usual harmonic bending potential. Tinker implements a Fourier bending term similar to that used by the Landis group's SHAPES force field. The parameters for specific Fourier angle terms are supplied via the ANGLEF parameter and keyword format. Note that a Fourier term will only be used for a particular angle if a corresponding harmonic angle term is not present in the parameter file.

We previously worked with the Anders Carlsson group at Washington University in St. Louis to add their transition metal ligand field term to Tinker. Support for this additional potential functional form is present in the distributed Tinker source code. We plan to develop energy routines and parameterization around alternative forms for handling transition metals, including the ligand field formulation proposed by Rob Deeth and coworkers.

7.8 Neighbor Methods for Nonbonded Terms

In addition to standard double loop methods, the Method of Lights is available to speed neighbor searching. This method based on taking intersections of sorted atom lists can be much faster for problems where the cutoff distance is significantly smaller than half the maximal cell dimension. The current version of Tinker does not implement the "neighbor list" schemes common to many other simulation packages.

7.9 Periodic Boundary Conditions

Both spherical cutoff images or replicates of a cell are supported by all Tinker programs that implement periodic boundary conditions. Whenever the cutoff distance is too large for the minimum image to be the only relevant neighbor (i.e., half the minimum box dimension for orthogonal cells), Tinker will automatically switch from the image formalism to use of replicated cells.

7.10 Distance Cutoffs for Energy Functions

Polynomial energy switching over a window is used for terms whose energy is small near the cutoff distance. For monopole electrostatic interactions, which are quite large in typical cutoff ranges, a two polynomial multiplicative-additive shifted energy switch unique to Tinker is applied. The Tinker method is similar in spirit to the force switching methods of Steinbach and Brooks, J. Comput. Chem., 15, 667-683 (1994). While the particle mesh Ewald method is preferred when periodic boundary conditions are present, Tinker's shifted energy switch with reasonable switching windows is quite satisfactory for most routine modeling problems. The shifted energy switch minimizes the perturbation of the energy and the gradient at the cutoff to acceptable levels. Problems should arise only if the property you wish to monitor is known to require explicit inclusion of long range components (i.e., calculation of the dielectric constant, etc.).

7.11 Ewald Summations Methods

Tinker contains a versions of the Ewald summation technique for inclusion of long range electrostatic interactions via periodic boundaries. The particle mesh Ewald (PME) method is available for simple charge-charge potentials, while regular Ewald is provided for polarizable atomic multipole interactions. The accuracy and speed of the regular and PME calculations is dependent on several interrelated parameters. For both methods, the Ewald coefficient and real-space cutoff distance must be set to reasonable and complementary values. Additional control variables for regular Ewald are the fractional coverage and number of vectors used in reciprocal space. For PME the additional control values are the B-spline order and charge grid dimensions. Complete control over all of these parameters is available via the Tinker keyfile mechanism. By default Tinker will select a set of parameters which provide a reasonable compromise between accuracy and speed, but these should be checked and modified as necessary for each individual system.

7.12 Continuum Solvation Models

Several alternative continuum solvation algorithms are contained within Tinker. All of these are accessed via the SOLVATE keyword and its modifiers. Two simple surface area methods are implemented: the ASP method of Eisenberg and McLachlan, and the SASA method from Scheraga's group. These methods are applicable to any of the standard Tinker force fields. Various schemes based on the generalized Born formalism are also available: the original 1990 numerical "Onion-shell" GB/SA method from Still's group, the 1997 analytical GB/SA method also due to Still, a pairwise descreening algorithm originally proposed by Hawkins, Cramer and Truhlar, and the analytical continuum solvation (ACE) method of Schaefer and Karplus. At present, the generalized Born methods should only be used with force fields having simple partial charge electrostatic interactions.

Some further comments are in order regarding the GB/SA-style solvation models. The Onion-shell model is provided mostly for comparison purposes. It uses an exact, analytical surface area calculation for the cavity term and the numerical scheme described in the original paper for the polarization term. This method is very slow, especially for large systems, and does not contain the contribution of the Born radii chain rule term to the first derivatives. We recommend its use only for single-point energy calculations. The other GB/SA methods ("analytical" Still, H-C-T pairwise descreening, and ACE) use an approximate cavity term based on Born radii, and do contain fully correct derivatives including the Born radii chain rule contribution. These methods all scale in CPU time with the square of the size of the system, and can be used with minimization, molecular dynamics and large molecules.

Finally, we note that the ACE solvation model should not be used with the current version of Tinker. The algorithm is fully implemented in the source code, but parameterization is not complete. As of late 2000, parameter values are only available in the literature for use of ACE with the older CHARMM19 force field. We plan to develop values for use with more modern all-atom force fields, and these will be incorporated into Tinker sometime in the future.

7.13 Polarizable Multipole Electrostatics

Atomic multipole electrostatics through the quadrupole moment is supported by the current version of Tinker, as is either mutual or direct dipole polarization. Ewald summation is available for inclusion of long range interactions. Calculations are implemented via a mixture of the CCP5 algorithms of W. Smith and the Applequist-Dykstra Cartesian polytensor method. At present analytical energy and Cartesian gradient code is provided.

The Tinker package allows intramolecular polarization to be treated via a version of the interaction damping scheme of Thole. To implement the Thole scheme, it is necessary to set all the mutual-1x-scale keywords to a value of one. The other polarization scaling keyword series, direct-1x-scale and polar-1x-scale, can be set independently to enable a wide variety of polarization models. In order to use an Applequist-style model without polarization damping, simply set the polar-damp keyword to zero.

7.14 Potential Energy Smoothing

Versions of our Potential Smoothing and Search (PSS) methodology have been implemented within Tinker. This methods belong to the same general family as Scheraga's Diffusion Equation Method, Straub's Gaussian Density Annealing, Shalloway's Packet Annealing and Verschelde's Effective Diffused Potential, but our algorithms reflect our own ongoing research in this area. In many ways the Tinker potential smoothing methods are the deterministic analog of stochastic simulated annealing. The PSS algorithms are very powerful, but are relatively new and are still undergoing modification, testing and calibration within our research group. This version of Tinker also includes a basin-hopping conformational scanning algorithm in the program SCAN which is particularly effective on smoothed potential surfaces.

7.15 Distance Geometry Metrization

A much improved and very fast random pairwise metrization scheme is available which allows good sampling during trial distance matrix generation without the usual structural anomalies and CPU constraints of other metrization procedures. An outline of the methodology and its application to NMR NOE-based structure refinement is described in the paper by Hodsdon, et al. in Journal of Molecular Biology, 264, 585-602 (1996). We have obtained good results with something like the keyword phrase trial-distribution pairwise 5, which performs 5% partial random pairwise metrization. For structures over several hundred atoms, a value less than 5 for the percentage of metrization should be fine.

CHAPTER

EIGHT

USE OF THE KEYWORD CONTROL FILE

8.1 Using Keywords to Control Tinker Calculations

This section contains detailed descriptions of the keyword parameters used to define or alter the course of a Tinker calculation. The keyword control file is optional in the sense that all of the Tinker programs will run in the absence of a keyfile and will simply use default values or query the user for needed information. However, the keywords allow use of a wide variety of algorithmic and procedural options, many of which are unavailable interactively.

Keywords are read from the keyword control file. All programs look first for a keyfile with the same base name as the input molecular system and ending in the extension .key. If this file does not exist, then Tinker tries to use a generic keyfile with the name Tinker.key and located in the same directory as the input system. If neither a system-specific nor a generic keyfile is present, Tinker will continue by using default values for keyword options and asking interactive questions as necessary.

Tinker searches the keyfile during the course of a calculation for relevant keywords that may be present. All keywords must appear as the first word on the line. Any blank space to the left of the keyword is ignored, and all contents of the keyfiles are case insensitive. Some keywords take modifiers; i.e., Tinker looks further on the same line for additional information, such as the value of some parameter related to the keyword. Modifier information is read in free format, but must be completely contained on the same line as the original keyword. Any lines contained in the keyfile which do not qualify as valid keyword lines are treated as comments and are simply ignored.

Several keywords take a list of integer values (atom numbers, for example) as modifiers. For these keywords the integers can simply be listed explicitly and separated by spaces, commas or tabs. If a range of numbers is desired, it can be specified by listing the negative of the first number of the range, followed by a separator and the last number of the range. For example, the keyword line ACTIVE 4 -9 17 23 could be used to add atoms 4, 9 through 17, and 23 to the set of active atoms during a Tinker calculation.

8.2 Keywords Grouped by Functionality

Listed below are the available Tinker keywords sorted into groups by general function. The section ends with an alphabetical list containing each individual keyword, along with a brief description of its action, possible keyword modifiers, and usage examples.

8.2.1 OUTPUT CONTROL KEYWORDS

ARCHIVE DEBUG DIGITS ECHO EXIT-PAUSE NOVERSION OVERWRITE PRINTOUT SAVE-CYCLE SAVE-FORCE SAVE-INDUCED SAVE-VELOCITY VERBOSE WRITEOUT

8.2.2 FORCE FIELD SELECTION KEYWORDS

FORCEFIELD PARAMETERS

8.2.3 POTENTIAL FUNCTION SELECTION KEYWORDS

ANGANGTERM ANGLETERM BONDTERM CHARGETERM CHGDPLTERM DIPOLETERM EXTRATERM IMPROPTERM IMPTORSTERM METALTERM MPOLETERM OPBENDTERM OPDISTTERM PITORSTERM POLARIZETERM RESTRAINTERM RXNFIELDTERM SOLVATETERM STRBNDTERM STRTORTERM TORSIONTERM TORTORTERM UREYTERM VDWTERM

8.2.4 POTENTIAL FUNCTION PARAMETER KEYWORDS

ANGANG ANGLE ANGLE3 ANGLE4 ANGLE5 ANGLEF ATOM BIOTYPE BOND BOND3 BOND4 BOND5 CHARGE DIPOLE DIPOLE3 DIPOLE4 DIPOLE5 ELECTNEG HBOND IMPROPER IMPTORS METAL MULTIPOLE OPBEND OPDIST PIATOM PIBOND PITORS POLARIZE SOLVATE STRBND STRTORS TORSION TORSION4 TORSION5 TORTOR UREYBRAD VDW VDW14 VDWPR

8.2.5 ENERGY UNIT CONVERSION KEYWORDS

ANGLEUNIT ANGANGUNIT BONDUNIT ELECTRIC IMPROPUNIT IMPTORUNIT OPBENDUNIT OPDISTUNIT PITORSUNIT STRBNDUNIT STRTORUNIT TORSIONUNIT TORTORUNIT UREYUNIT

8.2.6 LOCAL GEOMETRY FUNCTIONAL FORM KEYWORDS

ANGLE-CUBIC ANGLE-QUARTIC ANGLE-PENTIC ANGLE-SEXTIC BOND-CUBIC BOND-QUARTIC BONDTYPE MM2-STRBND PISYSTEM UREY-CUBIC UREY-QUARTIC

8.2.7 VAN DER WAALS FUNCTIONAL FORM KEYWORDS

A-EXPTERM B-EXPTERM C-EXPTERM DELTA-HALGREN EPSILONRULE GAMMA-HALGREN GAUSSTYPE RADIUSRULE RADIUSSIZE RADIUSTYPE VDW-12-SCALE VDW-13-SCALE VDW-14-SCALE VDW-15-SCALE VDW-CORRECTION VDWINDEX VDWTYPE

8.2.8 ELECTROSTATICS FUNCTIONAL FORM KEYWORDS

CHG-12-SCALE CHG-13-SCALE CHG-14-SCALE CHG-15-SCALE CHG-BUFFER DIELECTRIC DIRECT-11-SCALE DIRECT-12-SCALE DIRECT-13-SCALE DIRECT-14-SCALE MPOLE-12-SCALE MPOLE-13-SCALE MPOLE-14-SCALE MPOLE-15-SCALE MUTUAL-11-SCALE MUTUAL-12-SCALE MUTUAL-13-SCALE MUTUAL-14-SCALE POLAR-12-SCALE POLAR-13-SCALE POLAR-14-SCALE POLAR-15-SCALE POLAR-ASPC POLAR-EPS POLAR-SOR POLARIZATION REACTIONFIELD

8.2.9 NONBONDED CUTOFF KEYWORDS

CHG-CUTOFF CHG-TAPER CUTOFF DPL-CUTOFF DPL-TAPER HESS-CUTOFF LIGHTS MPOLE-CUTOFF MPOLE-TAPER NEIGHBOR-GROUPS NEUTRAL-GROUPS POLYMER-CUTOFF TAPER TRUNCATE VDW-CUTOFF VDW-TAPER

8.2.10 EWALD SUMMATION KEYWORDS

EWALD EWALD-ALPHA EWALD-BOUNDARY EWALD-CUTOFF PME-GRID PME-ORDER

8.2.11 CRYSTAL LATTICE & PERIODIC BOUNDARY KEYWORDS

A-AXIS B-AXIS C-AXIS ALPHA BETA GAMMA NO-SYMMETRY OCTAHEDRON SPACEGROUP X-AXIS Y-AXIS Z-AXIS

8.2.12 NEIGHBOR LIST KEYWORDS

CHG-LIST LIST-BUFFER MPOLE-LIST NEIGHBOR-LIST VDW-LIST

8.2.13 OPTIMIZATION KEYWORDS

ANGMAX CAPPA FCTMIN HGUESS INTMAX LBFGS-VECTORS MAXITER NEWHESS NEXTITER SLOPEMAX STEEPEST-DESCENT STEPMAX STEPMIN

8.2.14 MOLECULAR DYNAMICS KEYWORDS

BEEMAN-MIXING DEGREES-FREEDOM INTEGRATOR REMOVE-INERTIA

8.2.15 THERMOSTAT & BAROSTAT KEYWORDS

ANISO-PRESSURE BAROSTAT COLLISION COMPRESS FRICTION FRICTION-SCALING TAU-PRESSURE TAU-TEMPERATURE THERMOSTAT VOLUME-MOVE VOLUME-SCALE VOLUME-TRIAL

8.2.16 TRANSITION STATE KEYWORDS

DIVERGE GAMMAMIN REDUCE SADDLEPOINT

8.2.17 DISTANCE GEOMETRY KEYWORDS

TRIAL-DISTANCE TRIAL-DISTRIBUTION

8.2.18 VIBRATIONAL ANALYSIS KEYWORDS

IDUMP VIB-ROOTS VIB-TOLERANCE

8.2.19 IMPLICIT SOLVATION KEYWORDS

BORN-RADIUS GK-RADIUS GKC GKR SOLVENT-PRESSURE SURFACE-TENSION

8.2.20 POISSON-BOLTZMANN KEYWORDS

AGRID APBS-GRID BCFL CGCENT CGRID FGCENT FGRID ION MG-AUTO MG-MANUAL PB-RADIUS PDIE SDENS SDIE SMIN SRAD SRFM SWIN

8.2.21 MATHEMATICAL ALGORITHM KEYWORDS

FFT-PACKAGE RANDOMSEED

8.2.22 PARALLELIZATION KEYWORDS

OPENMP-THREADS

8.2.23 FREE ENERGY PERTURBATION KEYWORDS

CHG-LAMBDA DPL-LAMBDA LAMBDA LIGAND MPOLE-LAMBDA MUTATE POLAR-LAMBDA VDW-LAMBDA

8.2.24 PARTIAL STRUCTURE KEYWORDS

ACTIVE GROUP GROUP-INTER GROUP-INTRA GROUP-MOLECULE GROUP-SELECT INACTIVE

8.2.25 CONSTRAINT & RESTRAINT KEYWORDS

BASIN ENFORCE-CHIRALITY RATTLE RATTLE-DISTANCE RATTLE-EPS RATTLE-LINE RATTLE-ORIGIN RATTLE-PLANE RESTRAIN-ANGLE RESTRAIN-DISTANCE RESTRAIN-GROUPS RESTRAIN-POSITION RESTRAIN-TORSION SPHERE WALL

8.2.26 PARAMETER FITTING KEYWORDS

FIT-ANGLE FIT-BOND FIT-OPBEND FIT-STRBND FIT-TORSION FIT-UREY FIX-ANGLE FIX-BOND FIX-DIPOLE FIX-MONOPOLE FIX-OPBEND FIX-QUADRUPOLE FIX-STRBND FIX-TORSION FIX-UREY POTENTIAL-ATOMS POTENTIAL-FIT POTENTIAL-OFFSET POTENTIAL-SHELLS POTENTIAL-SPACING TARGET-DIPOLE TARGET-QUADRUPOLE

8.2.27 POTENTIAL SMOOTHING KEYWORDS

DEFORM DIFFUSE-CHARGE DIFFUSE-TORSION DIFFUSE-VDW SMOOTHING

8.3 Description of Individual Keywords

The following is an alphabetical list of the Tinker keywords along with a brief description of the action of each keyword and required or optional parameters that can be used to extend or modify each keyword. The format of possible modifiers, if any, is shown in brackets following each keyword.

A-AXIS [real] Sets the value of the a-axis length for a crystal unit cell, or, equivalently, the X-axis length for a periodic box. The length value in Angstroms is listed after the keyword.

A-EXPTERM [real] Sets the value of the "A" premultiplier term in the Buckingham van der Waals function, i.e., the value of A in the formula Evdw = epsilon * { A exp[-B(Ro/R)] - C (Ro/R)6 }.

ACTIVE [integer list] Sets the list of active atoms during a Tinker computation. Individual potential energy terms are computed when at least one atom involved in the term is active. For Cartesian space calculations, active atoms are those allowed to move. For torsional space calculations, rotations are allowed when all atoms on one side of the rotated bond are active. Multiple ACTIVE lines can be present in the keyfile and are treated cumulatively. On each line the keyword can be

followed by one or more atom numbers or atom ranges. The presence of any ACTIVE keyword overrides any INACTIVE keywords in the keyfile.

ALPHA [real] Sets the value of the alpha angle of a crystal unit cell, i.e., the angle between the b-axis and c-axis of a unit cell, or, equivalently, the angle between the Y-axis and Z-axis of a periodic box. The default value in the absence of the ALPHA keyword is 90 degrees.

ANGANG [1 integer & 3 reals] This keyword provides the values for a single angle-angle cross term potential parameter.

ANGANGTERM [NONE/ONLY] This keyword controls use of the angle-angle cross term potential energy. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

ANGANGUNIT [real] Sets the scale factor needed to convert the energy value computed by the angle-angle cross term potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default of (Pi/180)^2 = 0.0003046 is used, if the ANGANGUNIT keyword is not given in the force field parameter file or the keyfile.

ANGLE [3 integers & 4 reals] This keyword provides the values for a single bond angle bending parameter. The integer modifiers give the atom class numbers for the three kinds of atoms involved in the angle which is to be defined. The real number modifiers give the force constant value for the angle and up to three ideal bond angles in degrees. In most cases only one ideal bond angle is given, and that value is used for all occurrences of the specified bond angle. If all three ideal angles are given, the values apply when the central atom of the angle is attached to 0, 1 or 2 additional hydrogen atoms, respectively. This "hydrogen environment" option is provided to implement the corresponding feature of Allinger's MM force fields. The default units for the force constant are kcal/mole/radian2, but this can be controlled via the ANGLEUNIT keyword.

ANGLE-CUBIC [real] Sets the value of the cubic term in the Taylor series expansion form of the bond angle bending potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. This term multiplied by the angle bending energy unit conversion factor, the force constant, and the cube of the deviation of the bond angle from its ideal value gives the cubic contribution to the angle bending energy. The default value in the absence of the ANGLE-CUBIC keyword is zero; i.e., the cubic angle bending term is omitted.

ANGLE-PENTIC [real] Sets the value of the fifth power term in the Taylor series expansion form of the bond angle bending potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. This term multiplied by the angle bending energy unit conversion factor, the force constant, and the fifth power of the deviation of the bond angle from its ideal value gives the pentic contribution to the angle bending energy. The default value in the absence of the ANGLE-PENTIC keyword is zero; i.e., the pentic angle bending term is omitted.

ANGLE-QUARTIC [real] Sets the value of the quartic term in the Taylor series expansion form of the bond angle bending potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. This term multiplied by the angle bending energy unit conversion factor, the force constant, and the forth power of the deviation of the bond angle from its ideal value gives the quartic contribution to the angle bending energy. The default value in the absence of the ANGLE-QUARTIC keyword is zero; i.e., the quartic angle bending term is omitted.

ANGLE-SEXTIC [real] Sets the value of the sixth power term in the Taylor series expansion form of the bond angle bending potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. This term multiplied by the angle bending energy unit conversion factor, the force constant, and the sixth power of the deviation of the bond angle from its ideal value gives the sextic contribution to the angle bending energy. The default value in the absence of the ANGLE-SEXTIC keyword is zero; i.e., the sextic angle bending term is omitted.

ANGLE3 [3 integers & 4 reals] This keyword provides the values for a single bond angle bending parameter specific to atoms in 3-membered rings. The integer modifiers give the atom class numbers for the three kinds of atoms involved in the angle which is to be defined. The real number modifiers give the force constant value for the angle and up to three ideal bond angles in degrees. If all three ideal angles are given, the values apply when the central atom of the angle is attached to 0, 1 or 2 additional hydrogen atoms, respectively. The default units for the force constant are kcal/mole/radian ^ 2, but this can be controlled via the ANGLEUNIT keyword. If any ANGLE3 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special ANGLE3 parameters be given for all angles in 3-membered rings. In the absence of any ANGLE3 keywords, standard ANGLE parameters will be used for bonds in 3-membered rings.

ANGLE4 [3 integers & 4 reals] This keyword provides the values for a single bond angle bending parameter specific to atoms in 4-membered rings. The integer modifiers give the atom class numbers for the three kinds of atoms involved in the angle which is to be defined. The real number modifiers give the force constant value for the angle and up to three ideal bond angles in degrees. If all three ideal angles are given, the values apply when the central atom of the angle is attached to 0, 1 or 2 additional hydrogen atoms, respectively. The default units for the force constant are kcal/mole/radian ^ 2, but this can be controlled via the ANGLEUNIT keyword. If any ANGLE4 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special ANGLE4 parameters be given for all angles in 4-membered rings. In the absence of any ANGLE4 keywords, standard ANGLE parameters will be used for bonds in 4-membered rings.

ANGLE5 [3 integers & 4 reals] This keyword provides the values for a single bond angle bending parameter specific to atoms in 5-membered rings. The integer modifiers give the atom class numbers for the three kinds of atoms involved in the angle which is to be defined. The real number modifiers give the force constant value for the angle and up to three ideal bond angles in degrees. If all three ideal angles are given, the values apply when the central atom of the angle is attached to 0, 1 or 2 additional hydrogen atoms, respectively. The default units for the force constant are kcal/mole/radian ^ 2, but this can be controlled via the ANGLEUNIT keyword. If any ANGLE5 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special ANGLE5 parameters be given for all angles in 5-membered rings. In the absence of any ANGLE5 keywords, standard ANGLE parameters will be used for bonds in 5-membered rings.

ANGLEF [3 integers & 3 reals] This keyword provides the values for a single bond angle bending parameter for a SHAPES-style Fourier potential function. The integer modifiers give the atom class numbers for the three kinds of atoms involved in the angle which is to be defined. The real number modifiers give the force constant value for the angle, the angle shift in degrees, and the periodicity value. Note that the force constant should be given as the "harmonic" value and not the native Fourier value. The default units for the force constant are kcal/mole/radian ^ 2, but this can be controlled via the ANGLEUNIT keyword.

ANGLETERM [NONE/ONLY] This keyword controls use of the bond angle bending potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The

NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

ANGLEUNIT [real] Sets the scale factor needed to convert the energy value computed by the bond angle bending potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of (Pi/180)^2 = 0.0003046 is used, if the ANGLEUNIT keyword is not given in the force field parameter file or the keyfile.

ANGMAX [real] Set the maximum permissible angle between the current optimization search direction and the negative of the gradient direction. If this maximum angle value is exceeded, the optimization routine will note an error condition and may restart from the steepest descent direction. The default value in the absence of the ANGMAX keyword is usually 88 degrees for conjugate gradient methods and 180 degrees (i.e., disabled) for variable metric optimizations.

ANISO-PRESSURE This keyword invokes use of full anisotropic pressure during dynamics simulations. When using this option, the three axis lengths and axis angles vary separately in response to the pressure tensor. The default, in the absence of the keyword, is isotropic pressure based on the average of the diagonal of the pressure tensor.

ARCHIVE This keyword causes Tinker molecular dynamics-based programs to write trajectories directly to a single plain-text archive file with the .arc format. If an archive file already exists at the start of the calculation, then the newly generated trajectory is appended to the end of the existing file. The default in the absence of this keyword is to write the trajectory snapshots to consecutively numbered cycle files.

ATOM [2 integers, name, quoted string, integer, real & integer] This keyword provides the values needed to define a single force field atom type.

B-AXIS [real] Sets the value of the b-axis length for a crystal unit cell, or, equivalently, the Y-axis length for a periodic box. The length value in Angstroms is listed after the keyword. If the keyword is absent, the b-axis length is set equal to the a-axis length.

B-EXPTERM [real] Sets the value of the "B" exponential factor in the Buckingham van der Waals function, i.e., the value of B in the formula Evdw = epsilon * { A exp[-B(Ro/R)] - C (Ro/R)6 }.

BAROSTAT [BERENDSEN] This keyword selects a barostat algorithm for use during molecular dynamics. At present only one modifier is available, a Berendsen bath coupling method. The default in the absence of the BAROSTAT keyword is to use the BERENDSEN algorithm.

BASIN [2 reals] Presence of this keyword turns on a "basin" restraint potential function that serves to drive the system toward a compact structure. The actual function is a Gaussian of the form Ebasin = epsilon * A exp[-B R^2], summed over all pairs of atoms where R is the distance between atoms. The A and B values are the depth and width parameters given as modifiers to the BASIN keyword. This potential is currently used to control the degree of expansion during potential energy smooth procedures through the use of shallow, broad basins.

BETA [real] Sets the value of the ? angle of a crystal unit cell, i.e., the angle between the a-axis and c-axis of a unit cell, or, equivalently, the angle between the X-axis and Z-axis of a periodic box. The default value in the absence of the BETA keyword is to set the beta angle equal to the alpha angle as given by the keyword ALPHA.

BIOTYPE [integer, name, quoted string & integer] This keyword provides the values to define the correspondence between a single biopolymer atom type and its force field atom type.

BOND [2 integers & 2 reals] This keyword provides the values for a single bond stretching parameter. The integer modifiers give the atom class numbers for the two kinds of atoms involved in the bond which is to be defined. The real number modifiers give the force constant value for the bond and the ideal bond length in Angstroms. The default units for the force constant are kcal/mole/Ang^2, but this can be controlled via the BONDUNIT keyword.

BOND-CUBIC [real] Sets the value of the cubic term in the Taylor series expansion form of the bond stretching potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. This term multiplied by the bond stretching energy unit conversion factor, the force constant, and the cube of the deviation of the bond length from its ideal value gives the cubic contribution to the bond stretching energy. The default value in the absence of the BOND-CUBIC keyword is zero; i.e., the cubic bond stretching term is omitted.

BOND-QUARTIC [real] Sets the value of the quartic term in the Taylor series expansion form of the bond stretching potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. This term multiplied by the bond stretching energy unit conversion factor, the force constant, and the forth power of the deviation of the bond length from its ideal value gives the quartic contribution to the bond stretching energy. The default value in the absence of the BOND-QUARTIC keyword is zero; i.e., the quartic bond stretching term is omitted.

BOND3 [2 integers & 2 reals] This keyword provides the values for a single bond stretching parameter specific to atoms in 3-membered rings. The integer modifiers give the atom class numbers for the two kinds of atoms involved in the bond which is to be defined. The real number modifiers give the force constant value for the bond and the ideal bond length in Angstroms. The default units for the force constant are kcal/mole/Ang^2, but this can be controlled via the BONDUNIT keyword. If any BOND3 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special BOND3 parameters be given for all bonds in 3-membered rings. In the absence of any BOND3 keywords, standard BOND parameters will be used for bonds in 3-membered rings.

BOND4 [2 integers & 2 reals] This keyword provides the values for a single bond stretching parameter specific to atoms in 4-membered rings. The integer modifiers give the atom class numbers for the two kinds of atoms involved in the bond which is to be defined. The real number modifiers give the force constant value for the bond and the ideal bond length in Angstroms. The default units for the force constant are kcal/mole/Ang^2, but this can be controlled via the BONDUNIT keyword. If any BOND4 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special BOND4 parameters be given for all bonds in 4-membered rings. In the absence of any BOND4 keywords, standard BOND parameters will be used for bonds in 4-membered rings

BOND5 [2 integers & 2 reals] This keyword provides the values for a single bond stretching parameter specific to atoms in 5-membered rings. The integer modifiers give the atom class numbers for the two kinds of atoms involved in the bond which is to be defined. The real number modifiers give the force constant value for the bond and the ideal bond length in Angstroms. The default units for the force constant are kcal/mole/Ang^2, but this can be controlled via the BONDUNIT keyword. If any BOND5 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special BOND5 parameters be given for all bonds in 5-membered rings. In the absence of any BOND5 keywords, standard BOND parameters will be used for bonds

in 5-membered rings

BONDTERM [NONE/ONLY] This keyword controls use of the bond stretching potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

BONDTYPE [TAYLOR/MORSE/GAUSSIAN] Chooses the functional form of the bond stretching potential. The TAYLOR option selects a Taylor series expansion containing terms from harmonic through quartic. The MORSE option selects a Morse potential fit to the ideal bond length and stretching force constant parameter values. The GAUSSIAN option uses an inverted Gaussian with amplitude equal to the Morse bond dissociation energy and width set to reproduce the vibrational frequency of a harmonic potential. The default is to use the TAYLOR potential.

BONDUNIT [real] Sets the scale factor needed to convert the energy value computed by the bond stretching potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the BONDUNIT keyword is not given in the force field parameter file or the keyfile.

C-AXIS [real] Sets the value of the C-axis length for a crystal unit cell, or, equivalently, the Z-axis length for a periodic box. The length value in Angstroms is listed after the keyword. If the keyword is absent, the C-axis length is set equal to the A-axis length.

C-EXPTERM [real] Sets the value of the "C" dispersion multiplier in the Buckingham van der Waals function, i.e., the value of C in the formula Evdw = epsilon * { A exp[-B(Ro/R)] - C (Ro/R)6 }.

CAPPA [real] This keyword is used to set the normal termination criterion for the line search phase of Tinker optimization routines. The line search exits successfully if the ratio of the current gradient projection on the line to the projection at the start of the line search falls below the value of CAPPA. A default value of 0.1 is used in the absence of the CAPPA keyword.

CHARGE [1 integer & 1 real] This keyword provides a value for a single atomic partial charge electrostatic parameter. The integer modifier, if positive, gives the atom type number for which the charge parameter is to be defined. Note that charge parameters are given for atom types, not atom classes. If the integer modifier is negative, then the parameter value to follow applies only to the individual atom whose atom number is the negative of the modifier. The real number modifier gives the values of the atomic partial charge in electrons.

CHARGETERM [NONE/ONLY] This keyword controls use of the charge-charge potential energy term between pairs of atomic partial charges. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

CHG-12-SCALE [real] This keyword provides a multiplicative scale factor that is applied to charge-charge electrostatic interactions between 1-2 connected atoms, i.e., atoms that are directly bonded. The default value of 0.0 is used, if the CHG-12-SCALE keyword is not given in either the parameter file or the keyfile.

CHG-13-SCALE [real] This keyword provides a multiplicative scale factor that is applied to charge-charge electrostatic interactions between 1-3 connected atoms, i.e., atoms separated by two covalent bonds. The default value of 0.0 is used, if the CHG-13-SCALE keyword is not given in either the parameter file or the keyfile.

CHG-14-SCALE [real] This keyword provides a multiplicative scale factor that is applied to charge-charge electrostatic interactions between 1-4 connected atoms, i.e., atoms separated by three covalent bonds. The default value of 1.0 is used, if the CHG-14-SCALE keyword is not given in either the parameter file or the keyfile.

CHG-15-SCALE [real] This keyword provides a multiplicative scale factor that is applied to charge-charge electrostatic interactions between 1-5 connected atoms, i.e., atoms separated by four covalent bonds. The default value of 1.0 is used, if the CHG-15-SCALE keyword is not given in either the parameter file or the keyfile.

CHG-CUTOFF [real] Sets the cutoff distance value in Angstroms for charge-charge electrostatic potential energy interactions. The energy for any pair of sites beyond the cutoff distance will be set to zero. Other keywords can be used to select a smoothing scheme near the cutoff distance. The default cutoff distance in the absence of the CHG-CUTOFF keyword is infinite for nonperiodic systems and 9.0 for periodic systems.

CHG-TAPER [real] This keyword allows modification of the cutoff window for charge-charge electrostatic potential energy interactions. It is similar in form and action to the TAPER keyword, except that its value applies only to the charge-charge potential. The default value in the absence of the CHG-TAPER keyword is to begin the cutoff window at 0.65 of the corresponding cutoff distance.

CHGDPLTERM [NONE/ONLY] This keyword controls use of the charge-dipole potential energy term between atomic partial charges and bond dipoles. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

COLLISION [real] Sets the value of the random collision frequency used in the Andersen stochastic collision dynamics thermostat. The supplied value has units of fs-1 atom-1 and is multiplied internal to Tinker by the time step in fs and N^2/3 where N is the number of atoms. The default value used in the absence of the COLLISION keyword is 0.1 which is appropriate for many systems but may need adjustment to achieve adequate temperature control without perturbing the dynamics.

COMPRESS [real] Sets the value of the bulk solvent isothermal compressibility in 1/Atm for use during pressure computation and scaling in molecular dynamics computations. The default value used in the absence of the COMPRESS keyword is 0.000046, appropriate for water. This parameter serves as a scale factor for the Groningen-style pressure bath coupling time, and its exact value should not be of critical importance.

CUTOFF [real] Sets the cutoff distance value for all nonbonded potential energy interactions. The energy for any of the nonbonded potentials of a pair of sites beyond the cutoff distance will be set to zero. Other keywords can be used to select a smoothing scheme near the cutoff distance, or to apply different cutoff distances to various nonbonded energy terms.

DEBUG Turns on printing of detailed information and intermediate values throughout the progress of a Tinker computation; not recommended for use with large structures or full potential energy functions since a summary of every individual interaction will usually be output.

DEFORM [real] Sets the amount of diffusion equation-style smoothing that will be applied to the potential energy surface when using the SMOOTH force field. The real number option is equivalent to the "time" value in the original Piela, et al. formalism; the larger the value, the greater the smoothing. The default value is zero, meaning that no smoothing will be applied.

DEGREES-FREEDOM [integer] This keyword allows manual setting of the number of degrees of

freedom during a dynamics calculation. The integer modifier is used by thermostating methods and in other places as the number of degrees of freedom, overriding the value determined by the Tinker code at dynamics startup. In the absence of the keyword, the programs will automatically compute the correct value based on the number of atoms active during dynamics, bond or other constrains, and use of periodic boundary conditions.

DELTA-HALGREN [real] Sets the value of the delta parameter in Halgren's buffered 14-7 vdw potential energy functional form. In the absence of the DELTA-HALGREN keyword, a default value of 0.07 is used.

DIELECTRIC [real] Sets the value of the bulk dielectric constant used to damp all electrostatic interaction energies for any of the Tinker electrostatic potential functions. The default value is force field dependent, but is usually equal to 1.0 (for Allinger's MM force fields the default is 1.5).

DIFFUSE-CHARGE [real] This keyword is used during potential function smoothing procedures to specify the effective diffusion coefficient to be applied to the smoothed form of the Coulomb's Law charge-charge potential function. In the absence of the DIFFUSE-CHARGE keyword, a default value of 3.5 is used.

DIFFUSE-TORSION [real] This keyword is used during potential function smoothing procedures to specify the effective diffusion coefficient to be applied to the smoothed form of the torsion angle potential function. In the absence of the DIFFUSE-TORSION keyword, a default value of 0.0225 is used.

DIFFUSE-VDW [real] This keyword is used during potential function smoothing procedures to specify the effective diffusion coefficient to be applied to the smoothed Gaussian approximation to the Lennard-Jones van der Waals potential function. In the absence of the DIFFUSE-VDW keyword, a default value of 1.0 is used.

DIGITS [integer] This keyword controls the number of digits of precision output by Tinker in reporting potential energies and atomic coordinates. The allowed values for the integer modifier are 4, 6 and 8. Input values less than 4 will be set to 4, and those greater than 8 will be set to 8. Final energy values reported by most Tinker programs will contain the specified number of digits to the right of the decimal point. The number of decimal places to be output for atomic coordinates is generally two larger than the value of DIGITS. In the absence of the DIGITS keyword a default value of 4 is used, and energies will be reported to 4 decimal places with coordinates to 6 decimal places.

DIPOLE [2 integers & 2 reals] This keyword provides the values for a single bond dipole electrostatic parameter. The integer modifiers give the atom type numbers for the two kinds of atoms involved in the bond dipole which is to be defined. The real number modifiers give the value of the bond dipole in Debyes and the position of the dipole site along the bond. If the bond dipole value is positive, then the first of the two atom types is the positive end of the dipole. For a negative bond dipole value, the first atom type listed is negative. The position along the bond is an optional modifier that gives the postion of the dipole site as a fraction between the first atom type (position=0) and the second atom type (position=1). The default for the dipole position in the absence of a specified value is 0.5, placing the dipole at the midpoint of the bond.

DIPOLE3 [2 integers & 2 reals] This keyword provides the values for a single bond dipole electrostatic parameter specific to atoms in 3-membered rings. The integer modifiers give the atom type numbers for the two kinds of atoms involved in the bond dipole which is to be defined. The real number modifiers give the value of the bond dipole in Debyes and the position of the dipole site

along the bond. The default for the dipole position in the absence of a specified value is 0.5, placing the dipole at the midpoint of the bond. If any DIPOLE3 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special DIPOLE3 parameters be given for all bond dipoles in 3-membered rings. In the absence of any DIPOLE3 keywords, standard DIPOLE parameters will be used for bonds in 3-membered rings.

DIPOLE4 [2 integers & 2 reals] This keyword provides the values for a single bond dipole electrostatic parameter specific to atoms in 4-membered rings. The integer modifiers give the atom type numbers for the two kinds of atoms involved in the bond dipole which is to be defined. The real number modifiers give the value of the bond dipole in Debyes and the position of the dipole site along the bond. The default for the dipole position in the absence of a specified value is 0.5, placing the dipole at the midpoint of the bond. If any DIPOLE4 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special DIPOLE4 parameters be given for all bond dipoles in 4-membered rings. In the absence of any DIPOLE4 keywords, standard DIPOLE parameters will be used for bonds in 4-membered rings.

DIPOLE5 [2 integers & 2 reals] This keyword provides the values for a single bond dipole electrostatic parameter specific to atoms in 5-membered rings. The integer modifiers give the atom type numbers for the two kinds of atoms involved in the bond dipole which is to be defined. The real number modifiers give the value of the bond dipole in Debyes and the position of the dipole site along the bond. The default for the dipole position in the absence of a specified value is 0.5, placing the dipole at the midpoint of the bond. If any DIPOLE5 keywords are present, either in the master force field parameter file or the keyfile, then Tinker requires that special DIPOLE5 parameters be given for all bond dipoles in 5-membered rings. In the absence of any DIPOLE5 keywords, standard DIPOLE parameters will be used for bonds in 5-membered rings.

DIPOLETERM [NONE/ONLY] This keyword controls use of the dipole-dipole potential energy term between pairs of bond dipoles. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

DIRECT-11-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the permanent (direct) field due to atoms within a polarization group during an induced dipole calculation, i.e., atoms that are in the same polarization group as the atom being polarized. The default value of 0.0 is used, if the DIRECT-11-SCALE keyword is not given in either the parameter file or the keyfile.

DIRECT-12-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the permanent (direct) field due to atoms in 1-2 polarization groups during an induced dipole calculation, i.e., atoms that are in polarization groups directly connected to the group containing the atom being polarized. The default value of 0.0 is used, if the DIRECT-12-SCALE keyword is not given in either the parameter file or the keyfile.

DIRECT-13-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the permanent (direct) field due to atoms in 1-3 polarization groups during an induced dipole calculation, i.e., atoms that are in polarization groups separated by one group from the group containing the atom being polarized. The default value of 0.0 is used, if the DIRECT-13-SCALE keyword is not given in either the parameter file or the keyfile.

DIRECT-14-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the permanent (direct) field due to atoms in 1-4 polarization groups during an induced dipole

calculation, i.e., atoms that are in polarization groups separated by two groups from the group containing the atom being polarized. The default value of 1.0 is used, if the DIRECT-14-SCALE keyword is not given in either the parameter file or the keyfile.

DIVERGE [real] This keyword is used by the SADDLE program to set the maximum allowed value of the ratio of the gradient length along the path to the total gradient norm at the end of a cycle of minimization perpendicular to the path. If the value provided by the DIVERGE keyword is exceeded, then another cycle of maximization along the path is required. A default value of 0.005 is used in the absence of the DIVERGE keyword.

DPL-CUTOFF [real] Sets the cutoff distance value in Angstroms for bond dipole-bond dipole electrostatic potential energy interactions. The energy for any pair of bond dipole sites beyond the cutoff distance will be set to zero. Other keywords can be used to select a smoothing scheme near the cutoff distance. The default cutoff distance in the absence of the DPL-CUTOFF keyword is essentially infinite for nonperiodic systems and 10.0 for periodic systems.

DPL-TAPER [real] This keyword allows modification of the cutoff windows for bond dipole-bond dipole electrostatic potential energy interactions. It is similar in form and action to the TAPER keyword, except that its value applies only to the vdw potential. The default value in the absence of the DPL-TAPER keyword is to begin the cutoff window at 0.75 of the dipole cutoff distance.

ECHO [text string] The presence of this keyword causes whatever text follows it on the line to be copied directly to the output file. This keyword is also active in parameter files. It has no default value; if no text follows the ECHO keyword, a blank line is placed in the output file.

ELECTNEG [3 integers & 1 real] This keyword provides the values for a single electronegativity bond length correction parameter. The first two integer modifiers give the atom class numbers of the atoms involved in the bond to be corrected. The third integer modifier is the atom class of an electronegative atom. In the case of a primary correction, an atom of this third class must be directly bonded to an atom of the second atom class. For a secondary correction, the third class is one atom removed from an atom of the second class. The real number modifier is the value in Angstroms by which the original ideal bond length is to be corrected.

ENFORCE-CHIRALITY This keyword causes the chirality found at chiral tetravalent centers in the input structure to be maintained during Tinker calculations. The test for chirality is not exhaustive; two identical monovalent atoms connected to a center cause it to be marked as non-chiral, but large equivalent substituents are not detected. Trivalent "chiral" centers, for example the alpha carbon in united-atom protein structures, are not enforced as chiral.

EPSILONRULE [GEOMETRIC/ARITHMETIC/HARMONIC/HHG] This keyword selects the combining rule used to derive the ? value for van der Waals interactions. The default in the absence of the EPSILONRULE keyword is to use the GEOMETRIC mean of the individual epsilon values of the two atoms involved in the van der Waals interaction.

EWALD This keyword turns on the use of Ewald summation during computation of electrostatic interactions in periodic systems. In the current version of Tinker, regular Ewald is used for polarizable atomic multipoles, and smooth particle mesh Ewald (PME) is used for charge-charge interactions. Ewald summation is not available for interactions involving bond-centered dipoles. By default, in the absence of the EWALD keyword, distance-based cutoffs are used for electrostatic interactions.

EWALD-ALPHA [real] Sets the value of the Ewald coefficient which controls the width of the Gaussian screening charges during particle mesh Ewald summation. In the absence of the EWALD-

ALPHA keyword, a value is chosen which causes interactions outside the real-space cutoff to be below a fixed tolerance. For most standard applications of Ewald summation, the program default should be used.

EWALD-BOUNDARY This keyword invokes the use of insulating (ie, vacuum) boundary conditions during Ewald summation, corresponding to the media surrounding the system having a dielectric value of 1. The default in the absence of the EWALD-BOUNDARY keyword is to use conducting (ie, tinfoil) boundary conditions where the surrounding media is assumed to have an infinite dielectric value.

EWALD-CUTOFF [real] Sets the value in Angstroms of the real-space distance cutoff for use during Ewald summation. By default, in the absence of the EWALD-CUTOFF keyword, a value of 9.0 is used.

EXIT-PAUSE This keyword causes Tinker programs to pause and wait for a carriage return at the end of executation prior to returning control to the operating system. This is useful to keep the execution window open following termination on machines running Microsoft Windows or Apple MacOS. The default in the absence of the EXIT-PAUSE keyword, is to return control to the operating system immediately at program termination.

EXTRATERM [NONE/ONLY] This keyword controls use of the user defined extra potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

FCTMIN [real] This keyword sets a convergence criterion for successful completion of a Tinker optimization. If the value of the optimization objective function, typically the potential energy, falls below the value set by FCTMIN, then the optimization is deemed to have converged. The default value in the absence of the FCTMIN keyword is -1000000, effectively removing this criterion as a possible agent for termination.

FORCEFIELD [name] This keyword provides a name for the force field to be used in the current calculation. Its value is usually set in the master force field parameter file for the calculation (see the PARAMETERS keyword) instead of in the keyfile.

FRICTION [real] Sets the value of the frictional coefficient in 1/ps for use with stochastic dynamics. The default value used in the absence of the FRICTION keyword is 91.0, which is generally appropriate for water.

FRICTION-SCALING This keyword turns on the use of atomic surface area-based scaling of the frictional coefficient during stochastic dynamics. When in use, the coefficient for each atom is multiplied by that atom's fraction of exposed surface area. The default in the absence of the keyword is to omit the scaling and use the full coefficient value for each atom.

GAMMA [real] Sets the value of the gamma angle of a crystal unit cell, i.e., the angle between the a-axis and b-axis of a unit cell, or, equivalently, the angle between the X-axis and Y-axis of a periodic box. The default value in the absence of the GAMMA keyword is to set the gamma angle equal to the gamma angle as given by the keyword ALPHA.

GAMMA-HALGREN [real] Sets the value of the gamma parameter in Halgren's buffered 14-7 vdw potential energy functional form. In the absence of the GAMMA-HALGREN keyword, a default value of 0.12 is used.

GAMMAMIN [real] Sets the convergence target value for gamma during searches for maxima along the quadratic synchronous transit used by the SADDLE program. The value of gamma is the square of the ratio of the gradient projection along the path to the total gradient. A default value of 0.00001 is used in the absence of the GAMMAMIN keyword.

GAUSSTYPE [LJ-2/LJ-4/MM2-2/MM3-2/IN-PLACE] This keyword specifies the underlying vdw form that a Gaussian vdw approximation will attempt to fit as the number of terms to be used in a Gaussian approximation of the Lennard-Jones van der Waals potential. The text modifier gives the name of the functional form to be used. Thus LJ-2 as a modifier will result in a 2-Gaussian fit to a Lennard-Jones vdw potential. The GAUSSTYPE keyword only takes effect when VDWTYPE is set to GAUSSIAN. This keyword has no default value.

GROUP [integer, integer list] This keyword defines an atom group as a substructure within the full input molecular structure. The value of the first integer is the group number which must be in the range from 1 to the maximum number of allowed groups. The remaining intergers give the atom or atoms contained in this group as one or more atom numbers or ranges. Multiple keyword lines can be used to specify additional atoms in the same group. Note that an atom can only be in one group, the last group to which it is assigned is the one used.

GROUP-INTER This keyword assigns a value of 1.0 to all inter-group interactions and a value of 0.0 to all intra-group interactions. For example, combination with the GROUP-MOLECULE keyword provides for rigid-body calculations.

GROUP-INTRA This keyword assigns a value of 1.0 to all intra-group interactions and a value of 0.0 to all inter-group interactions.

GROUP-MOLECULE This keyword sets each individual molecule in the system to be a separate atom group, but does not assign weights to group-group interactions.

GROUP-SELECT [2 integers, real] This keyword gives the weight in the final potential energy of a specified set of intra- or intergroup interactions. The integer modifiers give the group numbers of the groups involved. If the two numbers are the same, then an intragroup set of interactions is specified. The real modifier gives the weight by which all energetic interactions in this set will be multiplied before incorporation into the final potential energy. If omitted as a keyword modifier, the weight will be set to 1.0 by default. If any SELECT-GROUP keywords are present, then any set of interactions not specified in a SELECT-GROUP keyword is given a zero weight. The default when no SELECT-GROUP keywords are specified is to use all intergroup interactions with a weight of 1.0 and to set all intragroup interactions to zero.

HBOND [2 integers & 2 reals] This keyword provides the values for the MM3-style directional hydrogen bonding parameters for a single pair of atoms. The integer modifiers give the pair of atom class numbers for which hydrogen bonding parameters are to be defined. The two real number modifiers give the values of the minimum energy contact distance in Angstroms and the well depth at the minimum distance in kcal/mole.

HESS-CUTOFF [real] This keyword defines a lower limit for significant Hessian matrix elements. During computation of the Hessian matrix of partial second derivatives, any matrix elements with absolute value below HESS-CUTOFF will be set to zero and omitted from the sparse matrix Hessian storage scheme used by Tinker. For most calculations, the default in the absence of this keyword is zero, i.e., all elements will be stored. For most Truncated Newton optimizations the Hessian cutoff will be chosen dynamically by the optimizer.

HGUESS [real] Sets an initial guess for the average value of the diagonal elements of the scaled inverse Hessian matrix used by the optimally conditioned variable metric optimization routine. A default value of 0.4 is used in the absence of the HGUESS keyword.

IMPROPER [4 integers & 2 reals] This keyword provides the values for a single CHARMM-style improper dihedral angle parameter.

IMPROPTERM [NONE/ONLY] This keyword controls use of the CHARMM-style improper dihedral angle potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

IMPROPUNIT [real] Sets the scale factor needed to convert the energy value computed by the CHARMM-style improper dihedral angle potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the IMPROPUNIT keyword is not given in the force field parameter file or the keyfile.

IMPTORS [4 integers & up to 3 real/real/integer triples] This keyword provides the values for a single AMBER-style improper torsional angle parameter. The first four integer modifiers give the atom class numbers for the atoms involved in the improper torsional angle to be defined. By convention, the third atom class of the four is the trigonal atom on which the improper torsion is centered. The torsional angle computed is literally that defined by the four atom classes in the order specified by the keyword. Each of the remaining triples of real/real/integer modifiers give the half-amplitude, phase offset in degrees and periodicity of a particular improper torsional term, respectively. Periodicities through 3-fold are allowed for improper torsional parameters.

IMPTORSTERM [NONE/ONLY] This keyword controls use of the AMBER-style improper torsional angle potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

IMPTORSUNIT [real] Sets the scale factor needed to convert the energy value computed by the AMBER-style improper torsional angle potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the IMPTORSUNIT keyword is not given in the force field parameter file or the keyfile.

INACTIVE [integer list] Sets the list of inactive atoms during a Tinker computation. Individual potential energy terms are not computed when all atoms involved in the term are inactive. For Cartesian space calculations, inactive atoms are not allowed to move. For torsional space calculations, rotations are not allowed when there are inactive atoms on both sides of the rotated bond. Multiple INACTIVE lines can be present in the keyfile, and on each line the keyword can be followed by one or more atom numbers or ranges. If any INACTIVE keys are found, all atoms are set to active except those listed on the INACTIVE lines. The ACTIVE keyword overrides all INACTIVE keywords found in the keyfile.

INTEGRATE [VERLET/BEEMAN/STOCHASTIC/RIGIDBODY] Chooses the integration method for propagation of dynamics trajectories. The keyword is followed on the same line by the name of the option. Standard Newtonian MD can be run using either VERLET for the Velocity Verlet method, or BEEMAN for the velocity form of Bernie Brook's "Better Beeman" method. A Velocity Verlet-based stochastic dynamics trajectory is selected by the STOCHASTIC modifier. A rigid-body dynamics

method is selected by the RIGIDBODY modifier. The default integration scheme is MD using the BEEMAN method.

INTMAX [integer] Sets the maximum number of interpolation cycles that will be allowed during the line search phase of an optimization. All gradient-based Tinker optimization routines use a common line search routine involving quadratic extrapolation and cubic interpolation. If the value of INTMAX is reached, an error status is set for the line search and the search is repeated with a much smaller initial step size. The default value in the absence of this keyword is optimization routine dependent, but is usually in the range 5 to 10.

LAMBDA [real] This keyword sets the value of the lambda path parameter for free energy perturbation calculations. The real number modifier specifies the position along the mutation path and must be a number in the range from 0 (initial state) to 1 (final state). The actual atoms involved in the mutation are given separately in individual MUTATE keyword lines.

LBFGS-VECTORS [integer] Sets the number of correction vectors used by the limited-memory L-BFGS optimization routine. The current maximum allowable value, and the default in the absence of the LBFGS-VECTORS keyword is 15.

LIGHTS This keyword turns on Method of Lights neighbor generation for the partial charge electrostatics and any of the van der Waals potentials. This method will yield identical energetic results to the standard double loop method. Method of Lights will be faster when the volume of a sphere with radius equal to the nonbond cutoff distance is significantly less than half the volume of the total system (i.e., the full molecular system, the crystal unit cell or the periodic box). It requires less storage than pairwise neighbor lists.

LIST-BUFFER [real] Sets the size of the neighbor list buffer in Angstroms. This value is added to the actual cutoff distance to determine which pairs will be kept on the neighbor list. The same buffer value is used for all neighbor lists. The default value in the absence of 2.0 is used in the absence of the LIST-BUFFER keyword.

MAXITER [integer] Sets the maximum number of minimization iterations that will be allowed for any Tinker program that uses any of the nonlinear optimization routines. The default value in the absence of this keyword is program dependent, but is always set to a very large number.

METAL This keyword provides the values for a single transition metal ligand field parameter. Note this keyword is present in the code, but not active in the current version of Tinker.

METALTERM [NONE/ONLY] This keyword controls use of the transition metal ligand field potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

MM2-STRBND This keyword switches the behavior of the stretch-bend potential function to match the formulation used by the MM2 force field. In MM2, stretching of bonds to attached hydrogen atoms is not including in computing the stretch-bend cross term energy. The default behavior in the absence of this keyword is to include stretching of attached hydrogen atoms as in the MM3 force field.

MPOLE-12-SCALE [real] This keyword provides a multiplicative scale factor that is applied to permanent atomic multipole electrostatic interactions between 1-2 connected atoms, i.e., atoms that are directly bonded. The default value of 0.0 is used, if the MPOLE-12-SCALE keyword is not given in either the parameter file or the keyfile.

MPOLE-13-SCALE [real] This keyword provides a multiplicative scale factor that is applied to permanent atomic multipole electrostatic interactions between 1-3 connected atoms, i.e., atoms separated by two covalent bonds. The default value of 0.0 is used, if the MPOLE-13-SCALE keyword is not given in either the parameter file or the keyfile.

MPOLE-14-SCALE [real] This keyword provides a multiplicative scale factor that is applied to permanent atomic multipole electrostatic interactions between 1-4 connected atoms, i.e., atoms separated by three covalent bonds. The default value of 1.0 is used, if the MPOLE-14-SCALE keyword is not given in either the parameter file or the keyfile.

MPOLE-15-SCALE [real] This keyword provides a multiplicative scale factor that is applied to permanent atomic multipole electrostatic interactions between 1-5 connected atoms, i.e., atoms separated by four covalent bonds. The default value of 1.0 is used, if the MPOLE-15-SCALE keyword is not given in either the parameter file or the keyfile.

MPOLE-CUTOFF [real] Sets the cutoff distance value in Angstroms for atomic multipole potential energy interactions. The energy for any pair of sites beyond the cutoff distance will be set to zero. Other keywords can be used to select a smoothing scheme near the cutoff distance. The default cutoff distance in the absence of the MPOLE-CUTOFF keyword is infinite for nonperiodic systems and 9.0 for periodic systems.

MPOLE-TAPER [real] This keyword allows modification of the cutoff window for atomic multipole potential energy interactions. It is similar in form and action to the TAPER keyword, except that its value applies only to the atomic multipole potential. The default value in the absence of the MPOLE-TAPER keyword is to begin the cutoff window at 0.65 of the corresponding cutoff distance.

MPOLETERM [NONE/ONLY] This keyword controls use of the atomic multipole electrostatics potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

MULTIPOLE [5 lines with: 3 or 4 integers & 1 real; 3 reals; 1 real; 2 reals; 3 reals] This keyword provides the values for a set of atomic multipole parameters at a single site. A complete keyword entry consists of three consequtive lines, the first line containing the MULTIPOLE keyword and the two following lines. The first line contains three integers which define the atom type on which the multipoles are centered, and the Z-axis and X-axis defining atom types for this center. The optional fourth integer contains the Y-axis defining atom type, and is only required for locally chiral multipole sites. The real number on the first line gives the monopole (atomic charge) in electrons. The second line contains three real numbers which give the X-, Y- and Z-components of the atomic dipole in electron-Ang. The final three lines, consisting of one, two and three real numbers give the upper triangle of the traceless atomic quadrupole tensor in electron-Ang^2.

MUTATE [3 integers] This keyword is used to specify atoms to be mutated during free energy perturbation calculations. The first integer modifier gives the atom number of an atom in the current system. The final two modifier values give the atom types corresponding the the lambda=0 and lambda=1 states of the specified atom.

MUTUAL-11-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the induced (mutual) field due to atoms within a polarization group during an induced dipole calculation, i.e., atoms that are in the same polarization group as the atom being polarized. The default value of 1.0 is used, if the MUTUAL-11-SCALE keyword is not given in either the parameter file or the keyfile.

MUTUAL-12-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the induced (mutual) field due to atoms in 1-2 polarization groups during an induced dipole calculation, i.e., atoms that are in polarization groups directly connected to the group containing the atom being polarized. The default value of 1.0 is used, if the MUTUAL-12-SCALE keyword is not given in either the parameter file or the keyfile.

MUTUAL-13-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the induced (mutual) field due to atoms in 1-3 polarization groups during an induced dipole calculation, i.e., atoms that are in polarization groups separated by one group from the group containing the atom being polarized. The default value of 1.0 is used, if the MUTUAL-13-SCALE keyword is not given in either the parameter file or the keyfile.

MUTUAL-14-SCALE [real] This keyword provides a multiplicative scale factor that is applied to the induced (mutual) field due to atoms in 1-4 polarization groups during an induced dipole calculation, i.e., atoms that are in polarization groups separated by two groups from the group containing the atom being polarized. The default value of 1.0 is used, if the MUTUAL-14-SCALE keyword is not given in either the parameter file or the keyfile.

NEIGHBOR-GROUPS This keyword causes the attached atom to be used in determining the charge-charge neighbor distance for all monovalent atoms in the molecular system. Its use causes all monovalent atoms to be treated the same as their attached atoms for purposes of including or scaling 1-2, 1-3 and 1-4 interactions. This option works only for the simple charge-charge electrostatic potential; it does not affect bond dipole or atomic multipole potentials. The NEIGHBOR-GROUPS scheme is similar to that used by some common force fields such as ENCAD.

NEIGHBOR-LIST This keyword turns on pairwise neighbor lists for partial charge electrostatics, polarize multipole electrostatics and any of the van der Waals potentials. This method will yield identical energetic results to the standard double loop method.

NEUTRAL-GROUPS This keyword causes the attached atom to be used in determining the charge-charge interaction cutoff distance for all monovalent atoms in the molecular system. Its use reduces cutoff discontinuities by avoiding splitting many of the largest charge separations found in typical molecules. Note that this keyword does not rigorously implement the usual concept of a "neutral group" as used in the literature with Amber/OPLS and other force fields. This option works only for the simple charge-charge electrostatic potential; it does not affect bond dipole or atomic multipole potentials.

NEWHESS [integer] Sets the number of algorithmic iterations between recomputation of the Hessian matrix. At present this keyword applies exclusively to optimizations using the Truncated Newton method. The default value in the absence of this keyword is 1, i.e., the Hessian is computed on every iteration.

NEXTITER [integer] Sets the iteration number to be used for the first iteration of the current computation. At present this keyword applies to optimization procedures where its use can effect convergence criteria, timing of restarts, and so forth. The default in the absence of this keyword is to take the initial iteration as iteration 1.

NOSE-MASS [real] This keyword sets the mass of particles making up the Nose-Hoover chain in that thermostating method. The default in the absence of the NOSE-MASS keyword is to use a mass of 0.1.

NOVERSION Turns off the use of version numbers appended to the end of filenames as the method

for generating filenames for updated copies of an existing file. The presence of this keyword results in direct use of input file names without a search for the highest available version, and requires the entry of specific output file names in many additional cases. By default, in the absence of this keyword, Tinker generates and attaches version numbers in a manner similar to the Digital OpenVMS operating system. For example, subsequent new versions of the file molecule.xyz would be written first to the file molecule.xyz_2, then to molecule.xyz_3, etc.

OCTAHEDRON Specifies that the periodic "box" is a truncated octahedron with maximal distance across the truncated octahedron as given by the A-AXIS keyword. All other unit cell and periodic box size-defining keywords are ignored if the OCTAHEDRON keyword is present.

OPBEND [2 integers & 1 real] This keyword provides the values for a single Allinger MM-style out-of-plane angle bending potential parameter. The first integer modifier is the atom class of the central trigonal atom and the second integer is the atom class of the out-of-plane atom. The real number modifier gives the force constant value for the out-of-plane angle. The default units for the force constant are kcal/mole/radian ^ 2, but this can be controlled via the OPBENDUNIT keyword.

OPBENDTERM [NONE/ONLY] This keyword controls use of the Allinger MM-style out-of-plane bending potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

OPBENDUNIT [real] Sets the scale factor needed to convert the energy value computed by the Allinger MM-style out-of-plane bending potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default of (Pi/180) ^ 2 = 0.0003046 is used, if the OPBENDUNIT keyword is not given in the force field parameter file or the keyfile.

OPDIST [4 integers & 1 real] This keyword provides the values for a single out-of-plane distance potential parameter. The first integer modifier is the atom class of the central trigonal atom and the three following integer modifiers are the atom classes of the three attached atoms. The real number modifier is the force constant for the harmonic function of the out-of-plane distance of the central atom. The default units for the force constant are kcal/mole/Ang ^ 2, but this can be controlled via the OPDISTUNIT keyword.

OPDISTTERM [NONE/ONLY] This keyword controls use of the out-of-plane distance potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

OPDISTUNIT [real] Sets the scale factor needed to convert the energy value computed by the out-of-plane distance potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the OPDISTUNIT keyword is not given in the force field parameter file or the keyfile.

OVERWRITE Causes Tinker programs, such as minimizations, that output intermediate coordinate sets to create a single disk file for the intermediate results which is successively overwritten with the new intermediate coordinates as they become available. This keyword is essentially the opposite of the SAVECYCLE keyword.

PARAMETERS [file name] Provides the name of the force field parameter file to be used for the current Tinker calculation. The standard file name extension for parameter files, .prm, is an op-

tional part of the file name modifier. The default in the absence of the PARAMETERS keyword is to look for a parameter file with the same base name as the molecular system and ending in the .prm extension. If a valid parameter file is not found, the user will asked to provide a file name interactively.

PIATOM [1 integer & 3 reals] This keyword provides the values for the pisystem MO potential parameters for a single atom class belonging to a pisystem.

PIBOND [2 integers & 2 reals] This keyword provides the values for the pisystem MO potential parameters for a single type of pisystem bond.

PIBOND4 [2 integers & 2 reals] This keyword provides the values for the pisystem MO potential parameters for a single type of pisystem bond contained in a 4-membered ring.

PIBOND5 [2 integers & 2 reals] This keyword provides the values for the pisystem MO potential parameters for a single type of pisystem bond contained in a 5-membered ring.

PISYSTEM [integer list] This keyword sets the atoms within a molecule that are part of a conjugated pi-orbital system. The keyword is followed on the same line by a list of atom numbers and/or atom ranges that constitute the pi-system. The Allinger MM force fields use this information to set up an MO calculation used to scale bond and torsion parameters involving pi-system atoms.

PITORS [2 integers & 1 real] This keyword provides the values for a single pi-orbital torsional angle potential parameter. The two integer modifiers give the atom class numbers for the atoms involved in the central bond of the torsional angle to be parameterized. The real modifier gives the value of the 2-fold Fourier amplitude for the torsional angle between p-orbitals centered on the defined bond atom classes. The default units for the stretch-torsion force constant can be controlled via the PITORSUNIT keyword.

PITORSTERM [NONE/ONLY] This keyword controls use of the pi-orbital torsional angle potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

PITORSUNIT [real] Sets the scale factor needed to convert the energy value computed by the pi-orbital torsional angle potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the PITORSUNIT keyword is not given in the force field parameter file or the keyfile.

PME-GRID [3 integers] This keyword sets the dimensions of the charge grid used during particle mesh Ewald summation. The three modifiers give the size along the X-, Y- and Z-axes, respectively. If either the Y- or Z-axis dimensions are omitted, then they are set equal to the X-axis dimension. The default in the absence of the PME-GRID keyword is to set the grid size along each axis to the smallest power of 2, 3 and/or 5 which is at least as large as 1.5 times the axis length in Angstoms. Note that the FFT used by PME is not restricted to, but is most efficient for, grid sizes which are powers of 2, 3 and/or 5.

PME-ORDER [integer] This keyword sets the order of the B-spline interpolation used during particle mesh Ewald summation. A default value of 8 is used in the absence of the PME-ORDER keyword.

POLAR-12-SCALE [real] This keyword provides a multiplicative scale factor that is applied to polarization interactions between 1-2 polarization groups, i.e., pairs of atoms that are in directly

connected polarization groups. The default value of 0.0 is used, if the POLAR-12-SCALE keyword is not given in either the parameter file or the keyfile.

POLAR-13-SCALE [real] This keyword provides a multiplicative scale factor that is applied to polarization interactions between 1-3 polarization groups, i.e., pairs of atoms that are in polarization groups separated by one other group. The default value of 0.0 is used, if the POLAR-13-SCALE keyword is not given in either the parameter file or the keyfile.

POLAR-14-SCALE [real] This keyword provides a multiplicative scale factor that is applied to polarization interactions between 1-4 polarization groups, i.e., pairs of atoms that are in polarization groups separated by two other groups. The default value of 1.0 is used, if the POLAR-14-SCALE keyword is not given in either the parameter file or the keyfile.

POLAR-15-SCALE [real] This keyword provides a multiplicative scale factor that is applied to polarization interactions between 1-5 polarization groups, i.e., pairs of atoms that are in polarization groups separated by three other groups. The default value of 1.0 is used, if the POLAR-15-SCALE keyword is not given in either the parameter file or the keyfile.

POLAR-DAMP [2 reals] Controls the strength of the damping function applied to induced dipoles and dipole polarization interaction energies. The first modifier sets the radius in Angstoms of a hypothetical atom with unit polarizability, while the second modifier sets the scale factor for the exponent of the damping function. The default values for the radius and the scale factor are 1.662 and 1.0, respectively. Damping is eliminated entirely by using this keyword to set the radius value to zero.

POLAR-EPS [real] This keyword sets the convergence criterion applied during computation of self-consistent induced dipoles. The calculation is deemed to have converged when the rms change in Debyes in the induced dipole at all polarizable sites is less than the value specified with this keyword. The default value in the absence of the keyword is 0.000001 Debyes.

POLAR-SOR [real] Sets a successive overrelaxation (SOR) factor for use in computation of induced atomic dipoles. Optimal values for this keyword will speed the induced dipole calculation, and poor values can result in convergence failure. The default value in the absence of the POLAR-SOR keyword is 0.7 which often a reasonable value when short-range intramolecular polarization is present. For models lacking intramolecular polarization, keyword values closer to 1.0 may be optimal.

POLARIZATION [DIRECT/MUTUAL] Selects between the use of direct and mutual dipole polarization for force fields that incorporate the polarization term. The DIRECT modifier avoids an iterative calculation by using only the permanent electric field in computation of induced dipoles. The MUTUAL option, which is the default in the absence of the POLARIZATION keyword, iterates the induced dipoles to self-consistency.

POLARIZE [1 integer, 1 real & up to 4 integers] This keyword provides the values for a single atomic dipole polarizability parameter. The integer modifier, if positive, gives the atom type number for which a polarizability parameter is to be defined. If the first integer modifier is negative, then the parameter value to follow applies only to the individual atom whose atom number is the negative of the modifier. The real number modifier gives the value of the dipole polarizability in Ang^3. The final integer modifiers list the atom type numbers of atoms directly bonded to the current atom and which will be considered to be part of the current atom's polarization group.

POLARIZETERM [NONE/ONLY] This keyword controls use of the atomic dipole polarization po-

tential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

POLYMER-CUTOFF [real] Sets the value of an additional cutoff parameter needed for infinite polymer systems. This value must be set to less than half the minimal periodic box dimension and should be greater than the largest possible interatomic distance that can be subject to scaling or exclusion as a local electrostatic or van der Waals interaction. The default in the absence of the POLYMER-CUTOFF keyword is 5.5 Angstroms.

PRINTOUT [integer] A general parameter for iterative procedures such as minimizations that sets the number of iterations between writes of status information to the standard output. The default value in the absence of the keyword is 1, i.e., the calculation status is given every iteration.

RADIUSRULE [ARITHMETIC/GEOMETRIC/CUBIC-MEAN] Sets the functional form of the radius combining rule for heteroatomic van der Waals potential energy interactions. The default in the absence of the RADIUSRULE keyword is to use the arithmetic mean combining rule to get radii for heteroatomic interactions.

RADIUSSIZE [RADIUS/DIAMETER] Determines whether the atom size values given in van der Waals parameters read from VDW keyword statements are interpreted as atomic radius or diameter values. The default in the absence of the RADIUSSIZE keyword is to assume that vdw size parameters are given as radius values.

RADIUSTYPE [R-MIN/SIGMA] Determines whether atom size values given in van der Waals parameters read from VDW keyword statements are interpreted as potential minimum (Rmin) or LJ-style sigma values. The default in the absence of the RADIUSTYPE keyword is to assume that vdw size parameters are given as Rmin values.

RANDOMSEED [integer] Followed by an integer value, this keyword sets the initial seed value for the random number generator used by Tinker. Setting RANDOMSEED to the same value as an earlier run will allow exact reproduction of the earlier calculation. (Note that this will not hold across different machine types.) RANDOMSEED should be set to a positive integer less than about 2 billion. In the absence of the RANDOMSEED keyword the seed is chosen "randomly" based upon the number of seconds that have elapsed in the current decade.

RATTLE [BONDS/ANGLES/DIATOMIC/TRIATOMIC/WATER] Invokes the rattle algorithm, a velocity version of shake, on portions of a molecular system during a molecular dynamic calculation. The RATTLE keyword can be followed by any of the modifiers shown, in which case all occurrences of the modifier species are constrained at ideal values taken from the bond and angle parameters of the force field in use. In the absence of any modifier, RATTLE constrains all bonds to hydrogen atoms at ideal bond lengths.

RATTLE-DISTANCE [2 integers] This keyword allows the use of a holonomic constraint between the two atoms whose numbers are specified on the keyword line. If the two atoms are involved in a covalent bond, then their distance is constrained to the ideal bond length from the force field. For nonbonded atoms, the rattle constraint is fixed at their distance in the input coordinate file.

RATTLE-LINE [integer]

RATTLE-ORIGIN [integer]

RATTLE-PLANE [integer]

REACTIONFIELD [2 reals & 1 integer] This keyword provides parameters needed for the reaction field potential energy calculation. The two real modifiers give the radius of the dielectric cavity and the ratio of the bulk dielectric outside the cavity to that inside the cavity. The integer modifier gives the number of terms in the reaction field summation to be used. In the absence of the REACTIONFIELD keyword, the default values are a cavity of radius 1000000 Ang, a dielectric ratio of 80 and use of only the first term of the reaction field summation.

REDUCE [real] Specifies the fraction between zero and one by which the path between starting and final conformational state will be shortened at each major cycle of the transition state location algorithm implemented by the SADDLE program. This causes the path endpoints to move up and out of the terminal structures toward the transition state region. In favorable cases, a nonzero value of the REDUCE modifier can speed convergence to the transition state. The default value in the absence of the REDUCE keyword is zero.

RESTRAIN-ANGLE [3 integers & 3 reals] This keyword implements a flat-welled harmonic potential that can be used to restrain the angle between three atoms to lie within a specified angle range. The integer modifiers contain the atom numbers of the three atoms whose angle is to be restrained. The first real modifier is the force constant in kcal/degree ^ 2 for the restraint. The last two real modifiers give the lower and upper bounds in degrees on the allowed angle values. If the angle lies between the lower and upper bounds, the restraint potential is zero. Outside the bounds, the harmonic restraint is applied. If the angle range modifiers are omitted, then the atoms are restrained to the angle found in the input structure. If the force constant is also omitted, a default value of 10.0 is used.

RESTRAIN-DISTANCE [2 integers & 3 reals] This keyword implements a flat-welled harmonic potential that can be used to restrain two atoms to lie within a specified distance range. The integer modifiers contain the atom numbers of the two atoms to be restrained. The first real modifier specifies the force constant in kcal/Ang ^ 2 for the restraint. The next two real modifiers give the lower and upper bounds in Angstroms on the allowed distance range. If the interatomic distance lies between these lower and upper bounds, the restraint potential is zero. Outside the bounds, the harmonic restraint is applied. If the distance range modifiers are omitted, then the atoms are restrained to the interatomic distance found in the input structure. If the force constant is also omitted, a default value of 100.0 is used.

RESTRAIN-GROUPS [2 integers & 3 reals] This keyword implements a flat-welled harmonic distance restraint between the centers-of-mass of two groups of atoms. The integer modifiers are the numbers of the two groups which must be defined separately via the GROUP keyword. The first real modifier is the force constant in kcal/Ang ^ 2 for the restraint. The last two real modifiers give the lower and upper bounds in Angstroms on the allowed intergroup center-of-mass distance values. If the distance range modifiers are omitted, then the groups are restrained to the distance found in the input structure. If the force constant is also omitted, a default value of 100.0 is used.

RESTRAIN-POSITION [1 integer & 5 reals] This keyword provides the ability to restrain an individual atom to a specified coordinate position. The initial integer modifier contains the atom number of the atom to be restrained. The first real modifier sets the force constant in kcal/Ang ^ 2 for the harmonic restraint potential. The next three real number modifiers give the X-, Y- and Z-coordinates to which the atom is tethered. The final real modifier defines a sphere around the specified coordinates within which the restraint value is zero. If the exclusion sphere radius is omitted, it is taken to be zero. If the coordinates are omitted, then the atom is restrained to the origin. If the force constant is also omitted, a default value of 100.0 is used.

RESTRAIN-TORSION [4 integers & 3 reals] This keyword implements a flat-welled harmonic potential that can be used to restrain the torsional angle between four atoms to lie within a specified angle range. The initial integer modifiers contains the atom numbers of the four atoms whose torsional angle, computed in the atom order listed, is to be restrained. The first real modifier gives a force constant in kcal/degree^2 for the restraint. The last two real modifiers give the lower and upper bounds in degrees on the allowed torsional angle values. The angle values given can wrap around across -180 and +180 degrees. Outside the allowed angle range, the harmonic restraint is applied. If the angle range modifiers are omitted, then the atoms are restrained to the torsional angle found in the input structure. If the force constant is also omitted, a default value of 1.0 is used.

RESTRAINTERM [NONE/ONLY] This keyword controls use of the restraint potential energy terms. In the absence of a modifying option, this keyword turns on use of these potentials. The NONE option turns off use of these potential energy terms. The ONLY option turns off all potential energy terms except for these terms.

RXNFIELDTERM [NONE/ONLY] This keyword controls use of the reaction field continuum solvation potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

SADDLEPOINT The presence of this keyword allows Newton-style second derivative-based optimization routine used by NEWTON, NEWTROT and other programs to converge to saddlepoints as well as minima on the potential surface. By default, in the absence of the SADDLEPOINT keyword, checks are applied that prevent convergence to stationary points having directions of negative curvature.

SAVE-CYCLE This keyword causes Tinker programs, such as minimizations, that output intermediate coordinate sets to save each successive set to the next consecutively numbered cycle file. The SAVE-CYCLE keyword is the opposite of the OVERWRITE keyword.

SAVE-FORCE This keyword causes Tinker molecular dynamics calculations to save the values of the force components on each atom to a separate cycle file. These files are written whenever the atomic coordinate snapshots are written during the dynamics run. Each atomic force file name contains as a suffix the cycle number followed by the letter f.

SAVE-INDUCED This keyword causes Tinker molecular dynamics calculations that involve polarizable atomic multipoles to save the values of the induced dipole components on each polarizable atom to a separate cycle file. These files are written whenever the atomic coordinate snapshots are written during the dynamics run. Each induced dipole file name contains as a suffix the cycle number followed by the letter u.

SAVE-VELOCITY This keyword causes Tinker molecular dynamics calculations to save the values of the velocity components on each atom to a separate cycle file. These files are written whenever the atomic coordinate snapshots are written during the dynamics run. Each velocity file name contains as a suffix the cycle number followed by the letter v.

SLOPEMAX [real] This keyword and its modifying value set the maximum allowed size of the ratio between the current and initial projected gradients during the line search phase of conjugate gradient or truncated Newton optimizations. If this ratio exceeds SLOPEMAX, then the initial step size is reduced by a factor of 10. The default value is usually set to 10000.0 when not specified via the SLOPEMAX keyword.

SMOOTHING [DEM/GDA/TOPHAT/STOPHAT] This keyword activates the potential energy smoothing methods. Several variations are available depending on the value of the modifier used: DEM= Diffusion Equation Method with a standard Gaussian kernel; GDA= Gaussian Density Annealing as proposed by the Straub group; TOPHAT= a local DEM-like method using a finite range "tophat" kernel; STOPHAT= shifted tophat smoothing.

SOLVATE [ASP/SASA/ONION/STILL/HCT/ACE/GBSA] Use of this keyword during energy calculations with any of the standard force fields turns on a continuum solvation free energy term. Several algorithms are available based on the modifier used: ASP= Eisenberg-McLachlan ASP method using the Wesson-Eisenberg vacuum-to-water parameters; SASA= the Ooi-Scheraga SASA method; ONION= the original 1990 Still "Onion-shell" GB/SA method; STILL= the 1997 analytical GB/SA method from Still's group; HCT= the pairwise descreening method of Hawkins, Cramer and Truhlar; ACE= the Analytical Continuum Electrostatics solvation method from the Karplus group; GBSA= equivalent to the STILL modifier. At present, GB/SA-style methods are only valid for force fields that use simple partial charge electrostatics.

SOLVATETERM [NONE/ONLY] This keyword controls use of the macroscopic solvation potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

SPACEGROUP [name] This keyword selects the space group to be used in manipulation of crystal unit cells and asymmetric units. The name option must be chosen from one of the following currently implemented space groups: P1, P1(-), P21, Cc, P21/a, P21/n, P21/c, C2/c, P212121, Pna21, Pn21a, Cmc21, Pccn, Pbcn, Pbca, P41, I41/a, P4(-)21c, P4(-)m2, R3c, P6(3)/mcm, Fm3(-)m, Im3(-)m.

SPHERE [4 reals, or 1 integer & 1 real] This keyword provides an alternative to the ACTIVE and INACTIVE keywords for specification of subsets of active atoms. If four real number modifiers are provided, the first three are taken as X-, Y- and Z-coordinates and the fourth is the radius of a sphere centered at these coordinates. In this case, all atoms within the sphere at the start of the calculation are active throughout the calculation, while all other atoms are inactive. Similarly if one integer and real number are given, an "active" sphere with radius set by the real is centered on the system atom with atom number given by the integer modifier. Multiple SPHERE keyword lines can be present in a single keyfile, and the list of active atoms specified by the spheres is cumulative.

STEEPEST-DESCENT This keyword forces the L-BFGS optimization routine used by the MINIMIZE program and other programs to perform steepest descent minimization. This option can be useful in conjunction with small step sizes for following minimum energy paths, but is generally inferior to the L-BFGS default for most optimization purposes.

STEPMAX [real] This keyword and its modifying value set the maximum size of an individual step during the line search phase of conjugate gradient or truncated Newton optimizations. The step size is computed as the norm of the vector of changes in parameters being optimized. The default value depends on the particular Tinker program, but is usually in the range from 1.0 to 5.0 when not specified via the STEPMAX keyword.

STEPMIN [real] This keyword and its modifying value set the minimum size of an individual step during the line search phase of conjugate gradient or truncated Newton optimizations. The step size is computed as the norm of the vector of changes in parameters being optimized. The default value is usually set to about 10-16 when not specified via the STEPMIN keyword.

STRBND [1 integer & 3 reals] This keyword provides the values for a single stretch-bend cross term potential parameter. The integer modifier gives the atom class number for the central atom of the bond angle involved in stretch-bend interactions. The real number modifiers give the force constant values to be used when the central atom of the angle is attached to 0, 1 or 2 additional hydrogen atoms, respectively. The default units for the stretch-bend force constant are kcal/mole/Ang-degree, but this can be controlled via the STRBNDUNIT keyword.

STRBNDTERM [NONE/ONLY] This keyword controls use of the bond stretching-angle bending cross term potential energy. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

STRBNDUNIT [real] Sets the scale factor needed to convert the energy value computed by the bond stretching-angle bending cross term potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the STRBNDUNIT keyword is not given in the force field parameter file or the keyfile.

STRTORS [2 integers & 1 real] This keyword provides the values for a single stretch-torsion cross term potential parameter. The two integer modifiers give the atom class numbers for the atoms involved in the central bond of the torsional angles to be parameterized. The real modifier gives the value of the stretch-torsion force constant for all torsional angles with the defined central bond atom classes. The default units for the stretch-torsion force constant can be controlled via the STRTORUNIT keyword.

STRTORTERM [NONE/ONLY] This keyword controls use of the bond stretching-torsional angle cross term potential energy. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

STRTORUNIT [real] Sets the scale factor needed to convert the energy value computed by the bond stretching-torsional angle cross term potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the STRTORUNIT keyword is not given in the force field parameter file or the keyfile.

TAPER [real] This keyword allows modification of the cutoff windows for nonbonded potential energy interactions. The nonbonded terms are smoothly reduced from their standard value at the beginning of the cutoff window to zero at the far end of the window. The far end of the window is specified via the CUTOFF keyword or its potential function specific variants. The modifier value supplied with the TAPER keyword sets the beginning of the cutoff window. The modifier can be given either as an absolute distance value in Angstroms, or as a fraction between zero and one of the CUTOFF distance. The default value in the absence of the TAPER keyword ranges from 0.65 to 0.9 of the CUTOFF distance depending on the type of potential function. The windows are implemented via polynomial-based switching functions, in some cases combined with energy shifting.

TAU-PRESSURE [real] Sets the coupling time in picoseconds for the Groningen-style pressure bath coupling used to control the system pressure during molecular dynamics calculations. A default value of 2.0 is used for TAU-PRESSURE in the absence of the keyword.

TAU-TEMPERATURE [real] Sets the coupling time in picoseconds for the Groningen-style tempera-

ture bath coupling used to control the system temperature during molecular dynamics calculations. A default value of 0.1 is used for TAU-TEMPERATURE in the absence of the keyword.

THERMOSTAT [BERENDSEN/ANDERSEN] This keyword selects a thermostat algorithm for use during molecular dynamics. Two modifiers are available, a Berendsen bath coupling method, and an Andersen stochastic collision method. The default in the absence of the THERMOSTAT keyword is to use the BERENDSEN algorithm.

TORSION [4 integers & up to 6 real/real/integer triples] This keyword provides the values for a single torsional angle parameter. The first four integer modifiers give the atom class numbers for the atoms involved in the torsional angle to be defined. Each of the remaining triples of real/real/integer modifiers give the amplitude, phase offset in degrees and periodicity of a particular torsional function term, respectively. Periodicities through 6-fold are allowed for torsional parameters.

TORSION4 [4 integers & up to 6 real/real/integer triples] This keyword provides the values for a single torsional angle parameter specific to atoms in 4-membered rings. The first four integer modifiers give the atom class numbers for the atoms involved in the torsional angle to be defined. The remaining triples of real number and integer modifiers operate as described above for the TORSION keyword.

TORSION5 [4 integers & up to 6 real/real/integer triples] This keyword provides the values for a single torsional angle parameter specific to atoms in 5-membered rings. The first four integer modifiers give the atom class numbers for the atoms involved in the torsional angle to be defined. The remaining triples of real number and integer modifiers operate as described above for the TORSION keyword.

TORSIONTERM [NONE/ONLY] This keyword controls use of the torsional angle potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

TORSIONUNIT [real] Sets the scale factor needed to convert the energy value computed by the torsional angle potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the TORSIONUNIT keyword is not given in the force field parameter file or the keyfile.

TORTOR [7 integers, then multiple lines of 2 integers and 1 real] This keyword is used to provide the values for a single torsion-torsion parameter. The first five integer modifiers give the atom class numbers for the atoms involved in the two adjacent torsional angles to be defined. The last two integer modifiers contain the number of data grid points that lie along each axis of the torsion-torsion map. For example, this value will be 13 for a 30 degree torsional angle spacing, i.e., 360/30 = 12, but 13 values are required since data values for -180 and +180 degrees must both be supplied. The subsequent lines contain the torsion-torsion map data as the integer values in degrees of each torsional angle and the target energy value in kcal/mole.

TORTORTERM [NONE/ONLY] This keyword controls use of the torsion-torsion potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

TORTORUNIT [real] Sets the scale factor needed to convert the energy value computed by the

torsion-torsion potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the TORTORUNIT keyword is not given in the force field parameter file or the keyfile.

TRIAL-DISTANCE [CLASSIC/RANDOM/TRICOR/HAVEL integer/PAIRWISE integer] Sets the method for selection of a trial distance matrix during distance geometry computations. The keyword takes a modifier that selects the method to be used. The HAVEL and PAIRWISE modifiers also require an additional integer value that specifies the number of atoms used in metrization and the percentage of metrization, respectively. The default in the absence of this keyword is to use the PAIRWISE method with 100 percent metrization. Further information on the various methods is given with the description of the Tinker distance geometry program.

TRIAL-DISTRIBUTION [real] Sets the initial value for the mean of the Gaussian distribution used to select trial distances between the lower and upper bounds during distance geometry computations. The value given must be between 0 and 1 which represent the lower and upper bounds respectively. This keyword is rarely needed since Tinker will usually be able to choose a reasonable value by default.

TRUNCATE Causes all distance-based nonbond energy cutoffs to be sharply truncated to an energy of zero at distances greater than the value set by the cutoff keyword(s) without use of any shifting, switching or smoothing schemes. At all distances within the cutoff sphere, the full interaction energy is computed.

UREY-CUBIC [real] Sets the value of the cubic term in the Taylor series expansion form of the Urey-Bradley potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. The default value in the absence of the UREY-CUBIC keyword is zero; i.e., the cubic Urey-Bradley term is omitted.

UREY-QUARTIC [real] Sets the value of the quartic term in the Taylor series expansion form of the Urey-Bradley potential energy. The real number modifier gives the value of the coefficient as a multiple of the quadratic coefficient. The default value in the absence of the UREY-QUARTIC keyword is zero; i.e., the quartic Urey-Bradley term is omitted.

UREYBRAD [3 integers & 2 reals] This keyword provides the values for a single Urey-Bradley cross term potential parameter. The integer modifiers give the atom class numbers for the three kinds of atoms involved in the angle for which a Urey-Bradley term is to be defined. The real number modifiers give the force constant value for the term and the target value for the 1-3 distance in Angstroms. The default units for the force constant are kcal/mole/Ang^2, but this can be controlled via the UREYUNIT keyword.

UREYTERM [NONE/ONLY] This keyword controls use of the Urey-Bradley potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

UREYUNIT [real] Sets the scale factor needed to convert the energy value computed by the Urey-Bradley potential into units of kcal/mole. The correct value is force field dependent and typically provided in the header of the master force field parameter file. The default value of 1.0 is used, if the UREYUNIT keyword is not given in the force field parameter file or the keyfile.

VDW [1 integer & 3 reals] This keyword provides values for a single van der Waals parameter. The integer modifier, if positive, gives the atom class number for which vdw parameters are to

be defined. Note that vdw parameters are given for atom classes, not atom types. The three real number modifiers give the values of the atom size in Angstroms, homoatomic well depth in kcal/mole, and an optional reduction factor for univalent atoms.

VDW-12-SCALE [real] This keyword provides a multiplicative scale factor that is applied to van der Waals potential interactions between 1-2 connected atoms, i.e., atoms that are directly bonded. The default value of 0.0 is used, if the VDW-12-SCALE keyword is not given in either the parameter file or the keyfile.

VDW-13-SCALE [real] This keyword provides a multiplicative scale factor that is applied to van der Waals potential interactions between 1-3 connected atoms, i.e., atoms separated by two covalent bonds. The default value of 0.0 is used, if the VDW-13-SCALE keyword is not given in either the parameter file or the keyfile.

VDW-14-SCALE [real] This keyword provides a multiplicative scale factor that is applied to van der Waals potential interactions between 1-4 connected atoms, i.e., atoms separated by three covalent bonds. The default value of 1.0 is used, if the VDW-14-SCALE keyword is not given in either the parameter file or the keyfile.

VDW-15-SCALE [real] This keyword provides a multiplicative scale factor that is applied to van der Waals potential interactions between 1-5 connected atoms, i.e., atoms separated by four covalent bonds. The default value of 1.0 is used, if the VDW-15-SCALE keyword is not given in either the parameter file or the keyfile.

VDW-CUTOFF [real] Sets the cutoff distance value in Angstroms for van der Waals potential energy interactions. The energy for any pair of van der Waals sites beyond the cutoff distance will be set to zero. Other keywords can be used to select a smoothing scheme near the cutoff distance. The default cutoff distance in the absence of the VDW-CUTOFF keyword is infinite for nonperiodic systems and 9.0 for periodic systems.

VDW-TAPER [real] This keyword allows modification of the cutoff windows for van der Waals potential energy interactions. It is similar in form and action to the TAPER keyword, except that its value applies only to the vdw potential. The default value in the absence of the VDW-TAPER keyword is to begin the cutoff window at 0.9 of the vdw cutoff distance.

VDW14 [1 integer & 2 reals] This keyword provides values for a single van der Waals parameter for use in 1-4 nonbonded interactions. The integer modifier, if positive, gives the atom class number for which vdw parameters are to be defined. Note that vdw parameters are given for atom classes, not atom types. The two real number modifiers give the values of the atom size in Angstroms and the homoatomic well depth in kcal/mole. Reduction factors, if used, are carried over from the VDW keyword for the same atom class.

VDWPR [2 integers & 2 reals] This keyword provides the values for the vdw parameters for a single special heteroatomic pair of atoms. The integer modifiers give the pair of atom class numbers for which special vdw parameters are to be defined. The two real number modifiers give the values of the minimum energy contact distance in Angstroms and the well depth at the minimum distance in kcal/mole.

VDWTERM [NONE/ONLY] This keyword controls use of the van der Waals repulsion-dispersion potential energy term. In the absence of a modifying option, this keyword turns on use of the potential. The NONE option turns off use of this potential energy term. The ONLY option turns off all potential energy terms except for this one.

VDWTYPE [LENNARD-JONES / BUCKINGHAM / BUFFERED-14-7 / MM3-HBOND / GAUSSIAN] Sets the functional form for the van der Waals potential energy term. The text modifier gives the name of the functional form to be used. The GAUSSIAN modifier value implements a two or four Gaussian fit to the corresponding Lennard-Jones function for use with potential energy smoothing schemes. The default in the absence of the VDWTYPE keyword is to use the standard two parameter Lennard-Jones function.

VERBOSE Turns on printing of secondary and informational output during a variety of Tinker computations; a subset of the more extensive output provided by the DEBUG keyword.

WALL [real] Sets the radius of a spherical boundary used to maintain droplet boundary conditions. The real modifier specifies the desired approximate radius of the droplet. In practice, an artificial van der Waals wall is constructed at a fixed buffer distance of 2.5 Angstroms outside the specified radius. The effect is that atoms which attempt to move outside the region defined by the droplet radius will be forced toward the center.

WRITEOUT [integer] A general parameter for iterative procedures such as minimizations that sets the number of iterations between writes of intermediate results (such as the current coordinates) to disk file(s). The default value in the absence of the keyword is 1, i.e., the intermediate results are written to file on every iteration. Whether successive intermediate results are saved to new files or replace previously written intermediate results is controlled by the OVERWRITE and SAVE-CYCLE keywords.

CHAPTER

NINE

ROUTINES & FUNCTIONS

The distribution version of the Tinker package contains over 1000 separate programs, subroutines and functions. This section contains a brief description of the purpose of most of these code units. Further information can be found in the comments located at the top of each source code file.

ACTIVE Subroutine

"active" sets the list of atoms that are used during each potential energy function calculation

ADDBASE Subroutine

"addbase" builds the Cartesian coordinates for a single nucleic acid base; coordinates are read from the Protein Data Bank file or found from internal coordinates, then atom types are assigned and connectivity data generated

ADDBOND Subroutine

"addbond" adds entries to the attached atoms list in order to generate a direct connection between two atoms

ADDIONS Subroutine

"addions" takes a currently defined solvated system and places ions, with removal of solvent molecules

ADDSIDE Subroutine

"addside" builds the Cartesian coordinates for a single amino acid side chain; coordinates are read from the Protein Data Bank file or found from internal coordinates, then atom types are assigned and connectivity data generated

ADJACENT Function

"adjacent" finds an atom connected to atom "i1" other than atom "i2"; if no such atom exists, then the closest atom in space is returned

ADJUST Subroutine

"adjust" modifies site bounds on the PME grid and returns an offset into the B-spline coefficient arrays

ALCHEMY Program

"alchemy" computes the free energy difference corresponding to a small perturbation by Boltzmann weighting the potential energy difference over a number of sample states; current version (incorrectly) considers the charge energy to be intermolecular in finding the perturbation energies

ALTELEC Subroutine

"altelec" constructs mutated electrostatic parameters based on the lambda mutation parameter "elambda"

ALTERPOL Subroutine

"alterpol" finds an output set of atomic multipole parameters which when used with an intergroup polarization model will give the same electrostatic potential around the molecule as the input set of multipole parameters with all atoms in one polarization group

ALTTORS Subroutine

"alttors" constructs mutated torsional parameters based on the lambda mutation parameter "tlambda"

AMBERYZE Subroutine

"amberyze" prints the force field parameters in a format needed by the Amber setup protocol for using AMOEBA within Amber

ANALYSIS Subroutine

"analysis" calls the series of routines needed to calculate the potential energy and perform energy partitioning analysis in terms of type of interaction or atom number

ANALYZE Program

"analyze" computes and displays the total potential energy; options are provided to display system and force field info, partition the energy by atom or by potential function type, show force field parameters by atom; output the large energy interactions and find electrostatic and inertial properties

ANGGUESS Function

"angguess" sets approximate angle bend force constants based on atom type and connected atoms

ANGLES Subroutine

"angles" finds the total number of bond angles and stores the atom numbers of the atoms defining each angle; for each angle to a trivalent central atom, the third bonded atom is stored for use in out-of-plane bending

ANNEAL Program

"anneal" performs a simulated annealing protocol by means of variable temperature molecular dynamics using either linear, exponential or sigmoidal cooling schedules

ANORM Function

"anorm" finds the norm (length) of a vector; used as a service routine by the Connolly surface area and volume computation

APBSEMPOLE Subroutine

APBSFINAL Subroutine

APBSINDUCE Subroutine

APBSINITIAL Subroutine

APBSNLINDUCE Subroutine

ARCHIVE Program

"archive" is a utility program for coordinate files which concatenates multiple coordinate sets into a new archive or performs any of several manipulations on an existing archive

ASET Subroutine

"aset" computes by recursion the A functions used in the evaluation of Slater-type (STO) overlap integrals

ATOMYZE Subroutine

"atomyze" prints the potential energy components broken down by atom and to a choice of precision

ATTACH Subroutine

"attach" generates lists of 1-3, 1-4 and 1-5 connectivities starting from the previously determined list of attached atoms (ie, 1-2 connectivity)

AUXINIT Subroutine

"auxinit" initializes auxiliary variables and settings for inertial extended Lagrangian induced dipole prediction

AVGPOLE Subroutine

"avgpole" condenses the number of multipole atom types based upon atoms with equivalent attachments and additional user specified sets of equivalent atoms

BAOAB Subroutine

"baoab" implements a constrained stochastic dynamics time step using the geodesic BAOAB scheme

BAR Program

"bar" computes the free energy, enthalpy and entropy difference between two states via Zwanzig free energy perturbation (FEP) and Bennett acceptance ratio (BAR) methods

BARCALC Subroutine

BASEFILE Subroutine

"basefile" extracts from an input filename the portion consisting of any directory name and the base filename; also reads any keyfile and sets information level values

BCUCOF Subroutine

"bcucof" determines the coefficient matrix needed for bicubic interpolation of a function, gradients and cross derivatives

BCUINT Subroutine

"bcuint" performs a bicubic interpolation of the function value on a 2D spline grid

BCUINT1 Subroutine

"bcuint1" performs a bicubic interpolation of the function value and gradient along the directions of a 2D spline grid

BCUINT2 Subroutine

"bcuint2" performs a bicubic interpolation of the function value, gradient and Hessian along the directions of a 2D spline grid

BEEMAN Subroutine

"beeman" performs a single molecular dynamics time step via the Beeman multistep recursion formula; uses original coefficients or Bernie Brooks' "Better Beeman" values

BETACF Function

"betacf" computes a rapidly convergent continued fraction needed by routine "betai" to evaluate the cumulative Beta distribution

BETAI Function

"betai" evaluates the cumulative Beta distribution function as the probability that a random variable from a distribution with Beta parameters "a" and "b" will be less than "x"

BIGBLOCK Subroutine

"bigblock" replicates the coordinates of a single unit cell to give a larger unit cell as a block of repeated units

BIOSORT Subroutine

"biosort" renumbers and formats biotype parameters used to convert biomolecular structure into force field atom types

BITORS Subroutine

"bitors" finds the total number of bitorsions as pairs of adjacent torsional angles, and the numbers of the five atoms defining each bitorsion

BMAX Function

"bmax" computes the maximum order of the B functions needed for evaluation of Slater-type (STO) overlap integrals

BNDERR Function

"bnderr" is the distance bound error function and derivatives; this version implements the original and Havel's normalized lower bound penalty, the normalized version is preferred when lower bounds are small (as with NMR NOE restraints), the original penalty is needed if large lower bounds are present

BNDGUESS Function

"bndguess" sets approximate bond stretch force constants based on atom type and connected atoms

BONDS Subroutine

"bonds" finds the total number of covalent bonds and stores the atom numbers of the atoms defining each bond

BORN Subroutine

"born" computes the Born radius of each atom for use with the various implicit solvation models

BORN1 Subroutine

"born1" computes derivatives of the Born radii with respect to atomic coordinates and increments total energy derivatives and virial components for potentials involving Born radii

BOUNDS Subroutine

"bounds" finds the center of mass of each molecule and translates any stray molecules back into the periodic box

BOXMIN Subroutine

"boxmin" uses minimization of valence and vdw potential energy to expand and refine a collection of solvent molecules in a periodic box

BOXMIN1 Function

"boxmin1" is a service routine that computes the energy and gradient during refinement of a periodic box

BSET Subroutine

"bset" computes by downward recursion the B functions used in the evaluation of Slater-type (STO) overlap integrals

BSPLGEN Subroutine

"bsplgen" gets B-spline coefficients and derivatives for a single PME atomic site along a particular direction

BSPLINE Subroutine

"bspline" calculates the coefficients for an n-th order B-spline approximation

BSPLINE_FILL Subroutine

"bspline_fill" finds B-spline coefficients and derivatives for PME atomic sites along the fractional coordinate axes

BSSTEP Subroutine

"bsstep" takes a single Bulirsch-Stoer step with monitoring of local truncation error to ensure accuracy

BUSSI Subroutine

"bussi" performs a single molecular dynamics time step via the Bussi-Parrinello isothermal-isobaric algorithm

CALENDAR Subroutine

"calendar" returns the current time as a set of integer values representing the year, month, day, hour, minute and second

CART_TO_FRAC Subroutine

"cart_to_frac" computes a transformation matrix to convert a multipole object in Cartesian coordinates to fractional

CBUILD Subroutine

"cbuild" performs a complete rebuild of the partial charge electrostatic neighbor list for all sites

CELLANG Subroutine

"cellang" computes atomic coordinates and unit cell parameters from fractional coordinates and lattice vectors

CELLATOM Subroutine

"cellatom" completes the addition of a symmetry related atom to a unit cell by updating the atom type and attachment arrays

CENTER Subroutine

"center" moves the weighted centroid of each coordinate set to the origin during least squares superposition

CERROR Subroutine

"cerror" is the error handling routine for the Connolly surface area and volume computation

CFFTB Subroutine

"cfftb" computes the backward complex discrete Fourier transform, the Fourier synthesis

CFFTB1 Subroutine

CFFTF Subroutine

"cfftf" computes the forward complex discrete Fourier transform, the Fourier analysis

CFFTF1 Subroutine

CFFTI Subroutine

"cffti" initializes arrays used in both forward and backward transforms; "ifac" is the prime factorization of "n", and "wsave" contains a tabulation of trigonometric functions

CFFTI1 Subroutine

CHIRER Function

"chirer" computes the chirality error and its derivatives with respect to atomic Cartesian coordinates as a sum the squares of deviations of chiral volumes from target values

CHKANGLE Subroutine

"chkangle" tests angles to be constrained for their presence in small rings and removes constraints that are redundant

CHKAROM Function

"chkatom" tests for the presence of a specified atom as a member of an aromatic ring

CHKPOLE Subroutine

"chkpole" inverts atomic multipole moments as necessary at sites with chiral local reference frame definitions

CHKRING Subroutine

"chkring" tests an atom or a set of connected atoms for their presence within a single 3- to 6-membered ring

CHKSIZE Subroutine

"chksize" computes a measure of overall global structural expansion or compaction from the number of excess upper or lower bounds matrix violations

CHKSOCKET Subroutine

CHKTREE Subroutine

"chktree" tests a minimum energy structure to see if it belongs to the correct progenitor in the existing map

CHKTTOR Subroutine

"chkttor" tests the attached atoms at a torsion-torsion central site and inverts the angle values if the site is chiral

CHKXYZ Subroutine

"chkxyz" finds any pairs of atoms with identical Cartesian coordinates, and prints a warning message

CHOLESKY Subroutine

"cholesky" uses a modified Cholesky method to solve the linear system Ax = b, returning "x" in "b"; "A" is a real symmetric positive definite matrix with its upper triangle (including the diagonal) stored by rows

CIRPLN Subroutine

"cirpln" determines the points of intersection between a specified circle and plane

CJKM Function

"cjkm" computes the coefficients of spherical harmonics expressed in prolate spheroidal coordinates

CLIGHT Subroutine

"clight" performs a complete rebuild of the partial charge pair neighbor list for all sites using the method of lights

CLIMBER Subroutine

CLIMBRGD Subroutine

CLIMBROT Subroutine

CLIMBTOR Subroutine

CLIMBXYZ Subroutine

CLIST Subroutine

"clist" performs an update or a complete rebuild of the nonbonded neighbor lists for partial charges

CLUSTER Subroutine

"cluster" gets the partitioning of the system into groups and stores a list of the group to which each atom belongs

CMP_TO_FMP Subroutine

"cmp_to_fmp" transforms the atomic multipoles from Cartesian to fractional coordinates

COLUMN Subroutine

"column" takes the off-diagonal Hessian elements stored as sparse rows and sets up indices to allow column access

COMMAND Subroutine

"command" uses the standard Unix-like iargc/getarg routines to get the number and values of arguments specified on the command line at program runtime

COMPRESS Subroutine

"compress" transfers only the non-buried tori from the temporary tori arrays to the final tori arrays

CONNECT Subroutine

"connect" sets up the attached atom arrays starting from a set of internal coordinates

CONNOLLY Subroutine

"connolly" uses the algorithms from the AMS/VAM programs of Michael Connolly to compute the analytical molecular surface area and volume of a collection of spherical atoms; thus it implements Fred Richards' molecular surface definition as a set of analytically defined spherical and toroidal polygons

CONNYZE Subroutine

"connyze" prints information onconnected atoms as lists of all atom pairs that are 1-2 through 1-5 interactions

CONTACT Subroutine

"contact" constructs the contact surface, cycles and convex faces

CONTROL Subroutine

"control" gets initial values for parameters that determine the output style and information level provided by Tinker

COORDS Subroutine

"coords" converts the three principal eigenvalues/vectors from the metric matrix into atomic coordinates, and calls a routine to compute the rms deviation from the bounds

CORRELATE Program

"correlate" computes the time correlation function of some user-supplied property from individual snapshot frames taken from a molecular dynamics or other trajectory

CREATEJVM Subroutine

CREATESERVER Subroutine

CREATESYSTEM Subroutine

CREATEUPDATE Subroutine

CRYSTAL Program

"crystal" is a utility which converts between fractional and Cartesian coordinates, and can generate full unit cells from asymmetric units

CSPLINE Subroutine

"cspline" computes the coefficients for a periodic interpolating cubic spline

CUTOFFS Subroutine

"cutoffs" initializes and stores spherical energy cutoff distance windows, Hessian element and Ewald sum cutoffs, and allocates pairwise neighbor lists

CYTSY Subroutine

"cytsy" solves a system of linear equations for a cyclically tridiagonal, symmetric, positive definite matrix

CYTSYP Subroutine

"cytsyp" finds the Cholesky factors of a cyclically tridiagonal symmetric, positive definite matrix given by two vectors

CYTSYS Subroutine

"cytsys" solves a cyclically tridiagonal linear system given the Cholesky factors

D1D2 Function

"d1d2" is a utility function used in computation of the reaction field recursive summation elements

DAMPDIR Subroutine

"dampdir" generates coefficients for the direct field damping function for powers of the interatomic distance

DAMPEWALD Subroutine

"dampewald" generates coefficients for Ewald error function damping for powers of the interatomic distance

DAMPMUT Subroutine

"dampmut" generates coefficients for the mutual field damping function for powers of the interatomic distance

DAMPPOLAR Subroutine

"damppolar" generates coefficients for the charge penetration damping function used for polarization interactions

DAMPPOLE Subroutine

"damppole" generates coefficients for the charge penetration damping function for powers of the interatomic distance

DAMPPOT Subroutine

"damppot" generates coefficients for the charge penetration damping function used for the electrostatic potential

DAMPREP Subroutine

"damprep" generates coefficients for the Pauli repulsion damping function for powers of the interatomic distance

DAMPTHOLE Subroutine

"dampthole" generates coefficients for the Thole damping function for powers of the interatomic distance

DBUILD Subroutine

"dbuild" performs a complete rebuild of the damped dispersion neighbor list for all sites

DEFLATE Subroutine

"deflate" uses the power method with deflation to compute the few largest eigenvalues and eigenvectors of a symmetric matrix

DELETE Subroutine

"delete" removes a specified atom from the Cartesian coordinates list and shifts the remaining atoms

DEPTH Function

DESTROYJVM Subroutine

DESTROYSERVER Subroutine

DFIELD0A Subroutine

"dfield0a" computes the direct electrostatic field due to permanent multipole moments via a double loop

DFIELD0B Subroutine

"dfield0b" computes the direct electrostatic field due to permanent multipole moments via a pair list

DFIELD0C Subroutine

"dfield0c" computes the mutual electrostatic field due to permanent multipole moments via Ewald summation

DFIELD0D Subroutine

"dfield0d" computes the direct electrostatic field due to permanent multipole moments for use with with generalized Kirkwood implicit solvation

DFIELD0E Subroutine

"dfield0e" computes the direct electrostatic field due to permanent multipole moments for use with in Poisson-Boltzmann

DFIELDI Subroutine

"dfieldi" computes the electrostatic field due to permanent multipole moments

DFTMOD Subroutine

"dftmod" computes the modulus of the discrete Fourier transform of "bsarray" and stores it in "bsmod"

DIAGBLK Subroutine

"diagblk" performs diagonalization of the Hessian for a block of atoms within a larger system

DIAGQ Subroutine

"diagq" is a matrix diagonalization routine which is derived from the classical given, housec, and eigen algorithms with several modifications to increase efficiency and accuracy

DIFFEQ Subroutine

"diffeq" performs the numerical integration of an ordinary differential equation using an adaptive stepsize method to solve the corresponding coupled first-order equations of the general form dyi/dx = f(x,y1,...,yn) for yi = y1,...,yn

DIFFUSE Program

"diffuse" finds the self-diffusion constant for a homogeneous liquid via the Einstein relation from a set of stored molecular dynamics frames; molecular centers of mass are unfolded and mean squared displacements are computed versus time separation

DIST2 Function

"dist2" finds the distance squared between two points; used as a service routine by the Connolly surface area and volume computation

DISTGEOM Program

"distgeom" uses a metric matrix distance geometry procedure to generate structures with interpoint distances that lie within specified bounds, with chiral centers that maintain chirality, and with torsional angles restrained to desired values; the user also has the ability to interactively inspect and alter the triangle smoothed bounds matrix prior to embedding

DLIGHT Subroutine

"dlight" performs a complete rebuild of the damped dispersion pair neighbor list for all sites using the method of lights

DLIST Subroutine

"dlist" performs an update or a complete rebuild of the nonbonded neighbor lists for damped dispersion sites

DMDUMP Subroutine

"dmdump" puts the distance matrix of the final structure into the upper half of a matrix, the distance of each atom to the centroid on the diagonal, and the individual terms of the bounds errors into the lower half of the matrix

DOCUMENT Program

"document" generates a formatted description of all the code modules or common blocks, an index of routines called by each source code module, a listing of all valid keywords, a list of include file dependencies as needed by a Unix-style Makefile, or a formatted force field parameter set summary

DOT Function

"dot" finds the dot product of two vectors

DSTMAT Subroutine

"dstmat" selects a distance matrix containing values between the previously smoothed upper and lower bounds; the distance values are chosen from uniform distributions, in a triangle correlated fashion, or using random partial metrization

DYNAMIC Program

"dynamic" computes a molecular or stochastic dynamics trajectory in one of the standard statistical mechanical ensembles and using any of several possible integration methods

EANGANG Subroutine

"eangang" calculates the angle-angle potential energy

EANGANG1 Subroutine

"eangang1" calculates the angle-angle potential energy and first derivatives with respect to Cartesian coordinates

EANGANG2 Subroutine

"eangang2" calculates the angle-angle potential energy second derivatives with respect to Cartesian coordinates using finite difference methods

EANGANG2A Subroutine

"eangang2a" calculates the angle-angle first derivatives for a single interaction with respect to Cartesian coordinates; used in computation of finite difference second derivatives

EANGANG3 Subroutine

"eangang3" calculates the angle-angle potential energy; also partitions the energy among the atoms

EANGLE Subroutine

"eangle" calculates the angle bending potential energy; projected in-plane angles at trigonal centers, special linear or Fourier angle bending terms are optionally used

EANGLE1 Subroutine

"eangle1" calculates the angle bending potential energy and the first derivatives with respect to Cartesian coordinates; projected in-plane angles at trigonal centers, special linear or Fourier angle bending terms are optionally used

EANGLE2 Subroutine

"eangle2" calculates second derivatives of the angle bending energy for a single atom using a mixture of analytical and finite difference methods; projected in-plane angles at trigonal centers, special linear or Fourier angle bending terms are optionally used

EANGLE2A Subroutine

"eangle2a" calculates bond angle bending potential energy second derivatives with respect to Cartesian coordinates

EANGLE2B Subroutine

"eangle2b" computes projected in-plane bending first derivatives for a single angle with respect to Cartesian coordinates; used in computation of finite difference second derivatives

EANGLE3 Subroutine

"eangle3" calculates the angle bending potential energy, also partitions the energy among the atoms; projected in-plane angles at trigonal centers, spceial linear or Fourier angle bending terms are optionally used

EANGTOR Subroutine

"eangtor" calculates the angle-torsion potential energy

EANGTOR1 Subroutine

"eangtor1" calculates the angle-torsion energy and first derivatives with respect to Cartesian coordinates

EANGTOR2 Subroutine

"eangtor2" calculates the angle-torsion potential energy second derivatives with respect to Cartesian coordinates

EANGTOR3 Subroutine

"eangtor3" calculates the angle-torsion potential energy; also partitions the energy terms among the atoms

EBOND Subroutine

"ebond" calculates the bond stretching energy

EBOND1 Subroutine

"ebond1" calculates the bond stretching energy and first derivatives with respect to Cartesian coordinates

EBOND2 Subroutine

"ebond2" calculates second derivatives of the bond stretching energy for a single atom at a time

EBOND3 Subroutine

"ebond3" calculates the bond stretching energy; also partitions the energy among the atoms

EBUCK Subroutine

"ebuck" calculates the Buckingham exp-6 van der Waals energy

EBUCK0A Subroutine

"ebuck0a" calculates the Buckingham exp-6 van der Waals energy using a pairwise double loop

EBUCK0B Subroutine

"ebuck0b" calculates the Buckingham exp-6 van der Waals energy using the method of lights

EBUCK0C Subroutine

"ebuck0c" calculates the Buckingham exp-6 van der Waals energy using a pairwise neighbor list

EBUCK0D Subroutine

"ebuck0d" calculates the Buckingham exp-6 van der Waals energy via a Gaussian approximation for potential energy smoothing

EBUCK1 Subroutine

"ebuck1" calculates the Buckingham exp-6 van der Waals energy and its first derivatives with respect to Cartesian coordinates

EBUCK1A Subroutine

"ebuck1a" calculates the Buckingham exp-6 van der Waals energy and its first derivatives using a pairwise double loop

EBUCK1B Subroutine

"ebuck1b" calculates the Buckingham exp-6 van der Waals energy and its first derivatives using the method of lights

EBUCK1C Subroutine

"ebuck1c" calculates the Buckingham exp-6 van der Waals energy and its first derivatives using a pairwise neighbor list

EBUCK1D Subroutine

"ebuck1d" calculates the Buckingham exp-6 van der Waals energy and its first derivatives via a Gaussian approximation for potential energy smoothing

EBUCK2 Subroutine

"ebuck2" calculates the Buckingham exp-6 van der Waals second derivatives for a single atom at a time

EBUCK2A Subroutine

"ebuck2a" calculates the Buckingham exp-6 van der Waals second derivatives using a double loop over relevant atom pairs

EBUCK2B Subroutine

"ebuck2b" calculates the Buckingham exp-6 van der Waals second derivatives via a Gaussian approximation for use with potential energy smoothing

EBUCK3 Subroutine

"ebuck3" calculates the Buckingham exp-6 van der Waals energy and partitions the energy among the atoms

EBUCK3A Subroutine

"ebuck3a" calculates the Buckingham exp-6 van der Waals energy and partitions the energy among the atoms using a pairwise double loop

EBUCK3B Subroutine

"ebuck3b" calculates the Buckingham exp-6 van der Waals energy and also partitions the energy among the atoms using the method of lights

EBUCK3C Subroutine

"ebuck3c" calculates the Buckingham exp-6 van der Waals energy and also partitions the energy among the atoms using a pairwise neighbor list

EBUCK3D Subroutine

"ebuck3d" calculates the Buckingham exp-6 van der Waals energy via a Gaussian approximation for potential energy smoothing

ECHARGE Subroutine

"echarge" calculates the charge-charge interaction energy

ECHARGE0A Subroutine

"echarge0a" calculates the charge-charge interaction energy using a pairwise double loop

ECHARGE0B Subroutine

"echarge0b" calculates the charge-charge interaction energy using the method of lights

ECHARGE0C Subroutine

"echarge0c" calculates the charge-charge interaction energy using a pairwise neighbor list

ECHARGE0D Subroutine

"echarge0d" calculates the charge-charge interaction energy using a particle mesh Ewald summation

ECHARGE0E Subroutine

"echarge0e" calculates the charge-charge interaction energy using a particle mesh Ewald summation and the method of lights

ECHARGE0F Subroutine

"echarge0f" calculates the charge-charge interaction energy using a particle mesh Ewald summation and a neighbor list

ECHARGE0G Subroutine

"echarge0g" calculates the charge-charge interaction energy for use with potential smoothing methods

ECHARGE1 Subroutine

"echarge1" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates

ECHARGE1A Subroutine

"echarge1a" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates using a pairwise double loop

ECHARGE1B Subroutine

"echarge1b" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates using the method of lights

ECHARGE1C Subroutine

"echarge1c" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates using a pairwise neighbor list

ECHARGE1D Subroutine

"echarge1d" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates using a particle mesh Ewald summation

ECHARGE1E Subroutine

"echarge1e" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates using a particle mesh Ewald summation and the method of lights

ECHARGE1F Subroutine

"echarge1f" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates using a particle mesh Ewald summation and a neighbor list

ECHARGE1G Subroutine

"echarge1g" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates for use with potential smoothing methods

ECHARGE2 Subroutine

"echarge2" calculates second derivatives of the charge-charge interaction energy for a single atom

ECHARGE2A Subroutine

"echarge2a" calculates second derivatives of the charge-charge interaction energy for a single atom using a pairwise loop

ECHARGE2B Subroutine

"echarge2b" calculates second derivatives of the charge-charge interaction energy for a single atom using a neighbor list

ECHARGE2C Subroutine

"echarge2c" calculates second derivatives of the reciprocal space charge-charge interaction energy for a single atom using a particle mesh Ewald summation via numerical differentiation

ECHARGE2D Subroutine

"echarge2d" calculates second derivatives of the real space charge-charge interaction energy for a single atom using a pairwise loop

ECHARGE2E Subroutine

"echarge2e" calculates second derivatives of the real space charge-charge interaction energy for a single atom using a pairwise neighbor list

ECHARGE2F Subroutine

"echarge2f" calculates second derivatives of the charge-charge interaction energy for a single atom for use with potential smoothing methods

ECHARGE2R Subroutine

"echarge2r" computes reciprocal space charge-charge first derivatives; used to get finite difference second derivatives

ECHARGE3 Subroutine

"echarge3" calculates the charge-charge interaction energy and partitions the energy among the atoms

ECHARGE3A Subroutine

"echarge3a" calculates the charge-charge interaction energy and partitions the energy among the atoms using a pairwise double loop

ECHARGE3B Subroutine

"echarge3b" calculates the charge-charge interaction energy and partitions the energy among the atoms using the method of lights

ECHARGE3C Subroutine

"echarge3c" calculates the charge-charge interaction energy and partitions the energy among the atoms using a pairwise neighbor list

ECHARGE3D Subroutine

"echarge3d" calculates the charge-charge interaction energy and partitions the energy among the atoms using a particle mesh Ewald summation

ECHARGE3E Subroutine

"echarge3e" calculates the charge-charge interaction energy and partitions the energy among the atoms using a particle mesh Ewald summation and the method of lights

ECHARGE3F Subroutine

"echarge3f" calculates the charge-charge interaction energy and partitions the energy among the atoms using a particle mesh Ewald summation and a pairwise neighbor list

ECHARGE3G Subroutine

"echarge3g" calculates the charge-charge interaction energy and partitions the energy among the atoms for use with potential smoothing methods

ECHGDPL Subroutine

"echgdpl" calculates the charge-dipole interaction energy

ECHGDPL1 Subroutine

"echgdpl1" calculates the charge-dipole interaction energy and first derivatives with respect to Cartesian coordinates

ECHGDPL2 Subroutine

"echgdpl2" calculates second derivatives of the charge-dipole interaction energy for a single atom

ECHGDPL3 Subroutine

"echgdpl3" calculates the charge-dipole interaction energy; also partitions the energy among the atoms

ECHGTRN Subroutine

"echgtrn" calculates the charge transfer potential energy

ECHGTRN0A Subroutine

"echgtrn0a" calculates the charge transfer interaction energy using a double loop

ECHGTRN0B Subroutine

"echgtrn0b" calculates the charge transfer interaction energy using the method of lights

ECHGTRN0C Subroutine

"echgtrn0c" calculates the charge transfer interaction energy using a neighbor list

ECHGTRN1 Subroutine

"echgtrn1" calculates the charge transfer energy and first derivatives with respect to Cartesian coordinates

ECHGTRN1A Subroutine

"echgtrn1a" calculates the charge transfer interaction energy and first derivatives using a double loop

ECHGTRN1B Subroutine

"echgtrn1b" calculates the charge transfer energy and first derivatives using a pairwise neighbor list

ECHGTRN2 Subroutine

"echgtrn2" calculates the second derivatives of the charge transfer energy using a double loop over relevant atom pairs

ECHGTRN3 Subroutine

"echgtrn3" calculates the charge transfer energy; also partitions the energy among the atoms

ECHGTRN3A Subroutine

"echgtrn3a" calculates the charge transfer interaction energy and also partitions the energy among the atoms using a pairwise double loop

ECHGTRN3B Subroutine

"echgtrn3b" calculates the charge transfer interaction energy and also partitions the energy among the atoms using the method of lights

ECHGTRN3C Subroutine

"echgtrn3c" calculates the charge transfer interaction energy and also partitions the energy among the atoms using a pairwise neighbor list

ECRECIP Subroutine

"ecrecip" evaluates the reciprocal space portion of the particle mesh Ewald energy due to partial charges

ECRECIP1 Subroutine

"ecrecip1" evaluates the reciprocal space portion of the particle mesh Ewald summation energy and gradient due to partial charges

EDIFF Subroutine

"ediff" calculates the energy of polarizing the vacuum induced dipoles to their SCRF polarized values

EDIFF1A Subroutine

"ediff1a" calculates the energy and derivatives of polarizing the vacuum induced dipoles to their SCRF polarized values using a double loop

EDIFF1B Subroutine

"ediff1b" calculates the energy and derivatives of polarizing the vacuum induced dipoles to their SCRF polarized values using a neighbor list

EDIFF3 Subroutine

"ediff3" calculates the energy of polarizing the vacuum induced dipoles to their generalized Kirkwood values with energy analysis

EDIPOLE Subroutine

"edipole" calculates the dipole-dipole interaction energy

EDIPOLE1 Subroutine

"edipole1" calculates the dipole-dipole interaction energy and first derivatives with respect to Cartesian coordinates

EDIPOLE2 Subroutine

"edipole2" calculates second derivatives of the dipole-dipole interaction energy for a single atom

EDIPOLE3 Subroutine

"edipole3" calculates the dipole-dipole interaction energy; also partitions the energy among the atoms

EDISP Subroutine

"edisp" calculates the damped dispersion potential energy

EDISP0A Subroutine

"edisp0a" calculates the damped dispersion potential energy using a pairwise double loop

EDISP0B Subroutine

"edisp0b" calculates the damped dispersion potential energy using a pairwise neighbor list

EDISP0C Subroutine

"edisp0c" calculates the dispersion interaction energy using particle mesh Ewald summation and a double loop

EDISP0D Subroutine

"edisp0d" calculates the dispersion interaction energy using particle mesh Ewald summation and a neighbor list

EDISP1 Subroutine

"edisp1" calculates the damped dispersion energy and first derivatives with respect to Cartesian coordinates

EDISP1A Subroutine

"edisp1a" calculates the damped dispersion energy and derivatives with respect to Cartesian coordinates using a pairwise double loop

EDISP1B Subroutine

"edisp1b" calculates the damped dispersion energy and derivatives with respect to Cartesian coordinates using a pairwise neighbor list

EDISP1C Subroutine

"edisp1c" calculates the damped dispersion energy and derivatives with respect to Cartesian coordinates using particle mesh Ewald summation and a double loop

EDISP1D Subroutine

"edisp1d" calculates the damped dispersion energy and derivatives with respect to Cartesian coordinates using particle mesh Ewald summation and a neighbor list

EDISP2 Subroutine

"edisp2" calculates the damped dispersion second derivatives for a single atom at a time

EDISP3 Subroutine

"edisp3" calculates the dispersion energy; also partitions the energy among the atoms

EDISP3A Subroutine

"edisp3a" calculates the dispersion potential energy and also partitions the energy among the atoms using a pairwise double loop

EDISP3B Subroutine

"edisp3b" calculates the damped dispersion potential energy and also partitions the energy among the atomsusing a pairwise neighbor list

EDISP3C Subroutine

"edisp3c" calculates the dispersion interaction energy using particle mesh Ewald summation and a double loop

EDISP3D Subroutine

"edisp3d" calculates the damped dispersion energy and analysis using particle mesh Ewald summation and a neighbor list

EDREAL0C Subroutine

"edreal0c" calculates the damped dispersion potential energy using a particle mesh Ewald sum and pairwise double loop

EDREAL0D Subroutine

"edreal0d" evaluated the real space portion of the damped dispersion energy using a neighbor list

EDREAL1C Subroutine

"edreal1c" evaluates the real space portion of the Ewald summation energy and gradient due to damped dispersion interactions via a double loop

EDREAL1D Subroutine

"edreal1d" evaluates the real space portion of the Ewald summation energy and gradient due to damped dispersion interactions via a neighbor list

EDREAL3C Subroutine

"edreal3c" calculates the real space portion of the damped dispersion energy and analysis using Ewald and a double loop

EDREAL3D Subroutine

"edreal3d" evaluated the real space portion of the damped dispersion energy and analysis using Ewald and a neighbor list

EDRECIP Subroutine

"edrecip" evaluates the reciprocal space portion of the particle mesh Ewald energy due to damped dispersion

EDRECIP1 Subroutine

"edrecip1" evaluates the reciprocal space portion of particle mesh Ewald energy and gradient due to damped dispersion

EGAUSS Subroutine

"egauss" calculates the Gaussian expansion van der Waals energy

EGAUSS0A Subroutine

"egauss0a" calculates the Gaussian expansion van der Waals energy using a pairwise double loop

EGAUSS0B Subroutine

"egauss0b" calculates the Gaussian expansion van der Waals energy using the method of lights

EGAUSS0C Subroutine

"egauss0c" calculates the Gaussian expansion van der Waals energy using a pairwise neighbor list

EGAUSS0D Subroutine

"egauss0d" calculates the Gaussian expansion van der Waals energy for use with potential energy smoothing

EGAUSS1 Subroutine

"egauss1" calculates the Gaussian expansion van der Waals interaction energy and its first derivatives with respect to Cartesian coordinates

EGAUSS1A Subroutine

"egauss1a" calculates the Gaussian expansion van der Waals interaction energy and its first derivatives using a pairwise double loop

EGAUSS1B Subroutine

"egauss1b" calculates the Gaussian expansion van der Waals energy and its first derivatives with respect to Cartesian coordinates using the method of lights

EGAUSS1C Subroutine

"egauss1c" calculates the Gaussian expansion van der Waals energy and its first derivatives with respect to Cartesian coordinates using a pairwise neighbor list

EGAUSS1D Subroutine

"egauss1d" calculates the Gaussian expansion van der Waals interaction energy and its first derivatives for use with potential energy smoothing

EGAUSS2 Subroutine

"egauss2" calculates the Gaussian expansion van der Waals second derivatives for a single atom at a time

EGAUSS2A Subroutine

"egauss2a" calculates the Gaussian expansion van der Waals second derivatives using a pairwise double loop

EGAUSS2B Subroutine

"egauss2b" calculates the Gaussian expansion van der Waals second derivatives for use with potential energy smoothing

EGAUSS3 Subroutine

"egauss3" calculates the Gaussian expansion van der Waals interaction energy and partitions the energy among the atoms

EGAUSS3A Subroutine

"egauss3a" calculates the Gaussian expansion van der Waals energy and partitions the energy among the atoms using a pairwise double loop

EGAUSS3B Subroutine

"egauss3b" calculates the Gaussian expansion van der Waals energy and partitions the energy among the atoms using the method of lights

EGAUSS3C Subroutine

"egauss3c" calculates the Gaussian expansion van der Waals energy and partitions the energy among the atoms using a pairwise neighbor list

EGAUSS3D Subroutine

"egauss3d" calculates the Gaussian expansion van der Waals interaction energy and partitions the energy among the atoms for use with potential energy smoothing

EGB0A Subroutine

"egb0a" calculates the generalized Born polarization energy for the GB/SA solvation models using a pairwise double loop

EGB0B Subroutine

"egb0b" calculates the generalized Born polarization energy for the GB/SA solvation models using a pairwise neighbor list

EGB0C Subroutine

"egb0c" calculates the generalized Born polarization energy for the GB/SA solvation models for use with potential smoothing methods via analogy to the smoothing of Coulomb's law

EGB1A Subroutine

"egb1a" calculates the generalized Born electrostatic energy and first derivatives of the GB/SA solvation models using a double loop

EGB1B Subroutine

"egb1b" calculates the generalized Born electrostatic energy and first derivatives of the GB/SA solvation models using a neighbor list

EGB1C Subroutine

"egb1c" calculates the generalized Born energy and first derivatives of the GB/SA solvation models for use with potential smoothing methods

EGB2A Subroutine

"egb2a" calculates second derivatives of the generalized Born energy term for the GB/SA solvation models

EGB2B Subroutine

"egb2b" calculates second derivatives of the generalized Born energy term for the GB/SA solvation models for use with potential smoothing methods

EGB3A Subroutine

"egb3a" calculates the generalized Born electrostatic energy for GB/SA solvation models using a pairwise double loop; also partitions the energy among the atoms

EGB3B Subroutine

"egb3b" calculates the generalized Born electrostatic energy for GB/SA solvation models using a pairwise neighbor list; also partitions the energy among the atoms

EGB3C Subroutine

"egb3c" calculates the generalized Born electrostatic energy for GB/SA solvation models for use with potential smoothing methods via analogy to the smoothing of Coulomb's law; also partitions the energy among the atoms

EGEOM Subroutine

"egeom" calculates the energy due to restraints on positions, distances, angles and torsions as well as Gaussian basin and spherical droplet restraints

EGEOM1 Subroutine

"egeom1" calculates the energy and first derivatives with respect to Cartesian coordinates due to restraints on positions, distances, angles and torsions as well as Gaussian basin and spherical droplet restraints

EGEOM2 Subroutine

"egeom2" calculates second derivatives of restraints on positions, distances, angles and torsions as well as Gaussian basin and spherical droplet restraints

EGEOM3 Subroutine

"egeom3" calculates the energy due to restraints on positions, distances, angles and torsions as well as Gaussian basin and droplet restraints; also partitions energy among the atoms

EGK Subroutine

"egk" calculates the generalized Kirkwood electrostatic solvation free energy for the GK/NP implicit solvation model

EGK0A Subroutine

"egk0a" calculates the electrostatic portion of the implicit solvation energy via the generalized Kirkwood model

EGK1 Subroutine

"egk1" calculates the implicit solvation energy and derivatives via the generalized Kirkwood plus nonpolar implicit solvation

EGK1A Subroutine

"egk1a" calculates the electrostatic portion of the implicit solvation energy and derivatives via the generalized Kirkwood model

EGK3 Subroutine

"egk3" calculates the generalized Kirkwood electrostatic energy for GK/NP solvation models; also partitions the energy among the atoms

EGK3A Subroutine

"egk3a" calculates the electrostatic portion of the implicit solvation energy via the generalized Kirkwood model; also partitions the energy among the atoms

EHAL Subroutine

"ehal" calculates the buffered 14-7 van der Waals energy

EHAL0A Subroutine

"ehal0a" calculates the buffered 14-7 van der Waals energy using a pairwise double loop

EHAL0B Subroutine

"ehal0b" calculates the buffered 14-7 van der Waals energy using the method of lights

EHAL0C Subroutine

"ehal0c" calculates the buffered 14-7 van der Waals energy using a pairwise neighbor list

EHAL1 Subroutine

"ehal1" calculates the buffered 14-7 van der Waals energy and its first derivatives with respect to Cartesian coordinates

EHAL1A Subroutine

"ehal1a" calculates the buffered 14-7 van der Waals energy and its first derivatives with respect to Cartesian coordinates using a pairwise double loop

EHAL1B Subroutine

"ehal1b" calculates the buffered 14-7 van der Waals energy and its first derivatives with respect to Cartesian coordinates using the method of lights

EHAL1C Subroutine

"ehal1c" calculates the buffered 14-7 van der Waals energy and its first derivatives with respect to Cartesian coordinates using a pairwise neighbor list

EHAL2 Subroutine

"ehal2" calculates the buffered 14-7 van der Waals second derivatives for a single atom at a time

EHAL3 Subroutine

"ehal3" calculates the buffered 14-7 van der Waals energy and partitions the energy among the atoms

EHAL3A Subroutine

"ehal3a" calculates the buffered 14-7 van der Waals energy and partitions the energy among the atoms using a pairwise double loop

EHAL3B Subroutine

"ehal3b" calculates the buffered 14-7 van der Waals energy and also partitions the energy among the atoms using the method of lights

EHAL3C Subroutine

"ehal3c" calculates the buffered 14-7 van der Waals energy and also partitions the energy among the atoms using a pairwise neighbor list

EHPMF Subroutine

"ehpmf" calculates the hydrophobic potential of mean force energy using a pairwise double loop

EHPMF1 Subroutine

"ehpmf1" calculates the hydrophobic potential of mean force energy and first derivatives using a pairwise double loop

EHPMF3 Subroutine

"ehpmf3" calculates the hydrophobic potential of mean force nonpolar energy; also partitions the energy among the atoms

EIGEN Subroutine

"eigen" uses the power method to compute the largest eigenvalues and eigenvectors of the metric matrix, "valid" is set true if the first three eigenvalues are positive

EIGENRGD Subroutine

EIGENROT Subroutine

EIGENROT Subroutine

EIGENTOR Subroutine

EIGENXYZ Subroutine

EIMPROP Subroutine

"eimprop" calculates the improper dihedral potential energy

EIMPROP1 Subroutine

"eimprop1" calculates improper dihedral energy and its first derivatives with respect to Cartesian coordinates

EIMPROP2 Subroutine

"eimprop2" calculates second derivatives of the improper dihedral angle energy for a single atom

EIMPROP3 Subroutine

"eimprop3" calculates the improper dihedral potential energy; also partitions the energy terms among the atoms

EIMPTOR Subroutine

"eimptor" calculates the improper torsion potential energy

EIMPTOR1 Subroutine

"eimptor1" calculates improper torsion energy and its first derivatives with respect to Cartesian coordinates

EIMPTOR2 Subroutine

"eimptor2" calculates second derivatives of the improper torsion energy for a single atom

EIMPTOR3 Subroutine

"eimptor3" calculates the improper torsion potential energy; also partitions the energy terms among the atoms

ELJ Subroutine

"elj" calculates the Lennard-Jones 6-12 van der Waals energy

ELJ0A Subroutine

"elj0a" calculates the Lennard-Jones 6-12 van der Waals energy using a pairwise double loop

ELJ0B Subroutine

"elj0b" calculates the Lennard-Jones 6-12 van der Waals energy using the method of lights

ELJ0C Subroutine

"elj0c" calculates the Lennard-Jones 6-12 van der Waals energy using a pairwise neighbor list

ELJ0D Subroutine

"elj0d" calculates the Lennard-Jones 6-12 van der Waals energy via a Gaussian approximation for potential energy smoothing

ELJ0E Subroutine

"elj0e" calculates the Lennard-Jones 6-12 van der Waals energy for use with stophat potential energy smoothing

ELJ1 Subroutine

"elj1" calculates the Lennard-Jones 6-12 van der Waals energy and its first derivatives with respect to Cartesian coordinates

ELJ1A Subroutine

"elj1a" calculates the Lennard-Jones 6-12 van der Waals energy and its first derivatives using a pairwise double loop

ELJ1B Subroutine

"elj1b" calculates the Lennard-Jones 6-12 van der Waals energy and its first derivatives using the method of lights

ELJ1C Subroutine

"elj1c" calculates the Lennard-Jones 12-6 van der Waals energy and its first derivatives using a pairwise neighbor list

ELJ1D Subroutine

"elj1d" calculates the Lennard-Jones 6-12 van der Waals energy and its first derivatives via a Gaussian approximation for potential energy smoothing

ELJ1E Subroutine

"elj1e" calculates the van der Waals interaction energy and its first derivatives for use with stophat potential energy smoothing

ELJ2 Subroutine

"elj2" calculates the Lennard-Jones 6-12 van der Waals second derivatives for a single atom at a time

ELJ2A Subroutine

"elj2a" calculates the Lennard-Jones 6-12 van der Waals second derivatives using a double loop over relevant atom pairs

ELJ2B Subroutine

"elj2b" calculates the Lennard-Jones 6-12 van der Waals second derivatives via a Gaussian approximation for use with potential energy smoothing

ELJ2C Subroutine

"elj2c" calculates the Lennard-Jones 6-12 van der Waals second derivatives for use with stophat potential energy smoothing

ELJ3 Subroutine

"elj3" calculates the Lennard-Jones 6-12 van der Waals energy and also partitions the energy among the atoms

ELJ3A Subroutine

"elj3a" calculates the Lennard-Jones 6-12 van der Waals energy and also partitions the energy among the atoms using a pairwise double loop

ELJ3B Subroutine

"elj3b" calculates the Lennard-Jones 6-12 van der Waals energy and also partitions the energy among the atoms using the method of lights

ELJ3C Subroutine

"elj3c" calculates the Lennard-Jones van der Waals energy and also partitions the energy among the atoms using a pairwise neighbor list

ELJ3D Subroutine

"elj3d" calculates the Lennard-Jones 6-12 van der Waals energy and also partitions the energy among the atoms via a Gaussian approximation for potential energy smoothing

ELJ3E Subroutine

"elj3e" calculates the Lennard-Jones 6-12 van der Waals energy and also partitions the energy among the atoms for use with stophat potential energy smoothing

EMBED Subroutine

"embed" is a distance geometry routine patterned after the ideas of Gordon Crippen, Irwin Kuntz and Tim Havel; it takes as input a set of upper and lower bounds on the interpoint distances, chirality restraints and torsional restraints, and attempts to generate a set of coordinates that satisfy the input bounds and restraints

EMETAL Subroutine

"emetal" calculates the transition metal ligand field energy

EMETAL1 Subroutine

"emetal1" calculates the transition metal ligand field energy and its first derivatives with respect to Cartesian coordinates

EMETAL2 Subroutine

"emetal2" calculates the transition metal ligand field second derivatives for a single atom at a time

EMETAL3 Subroutine

"emetal3" calculates the transition metal ligand field energy and also partitions the energy among the atoms

EMM3HB Subroutine

"emm3hb" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy

EMM3HB0A Subroutine

"emm3hb0a" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy using a pairwise double loop

EMM3HB0B Subroutine

"emm3hb0b" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy using the method of lights

EMM3HB0C Subroutine

"emm3hb0c" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy using a pairwise neighbor list

EMM3HB1 Subroutine

"emm3hb1" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy with respect to Cartesian coordinates

EMM3HB1A Subroutine

"emm3hb1a" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy with respect to Cartesian coordinates using a pairwise double loop

EMM3HB1B Subroutine

"emm3hb1b" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy with respect to Cartesian coordinates using the method of lights

EMM3HB1C Subroutine

"emm3hb1c" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy with respect to Cartesian coordinates using a pairwise neighbor list

EMM3HB2 Subroutine

"emm3hb2" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding second derivatives for a single atom at a time

EMM3HB3 Subroutine

"emm3hb3" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy, and partitions the energy among the atoms

EMM3HB3A Subroutine

"emm3hb3" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy, and partitions the energy among the atoms

EMM3HB3B Subroutine

"emm3hb3b" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy using the method of lights

EMM3HB3C Subroutine

"emm3hb3c" calculates the MM3 exp-6 van der Waals and directional charge transfer hydrogen bonding energy using a pairwise neighbor list

EMPOLE Subroutine

"empole" calculates the electrostatic energy due to atomic multipole interactions

EMPOLE0A Subroutine

"empole0a" calculates the atomic multipole interaction energy using a double loop

EMPOLE0B Subroutine

"empole0b" calculates the atomic multipole interaction energy using a neighbor list

EMPOLE0C Subroutine

"empole0c" calculates the atomic multipole interaction energy using particle mesh Ewald summation and a double loop

EMPOLE0D Subroutine

"empole0d" calculates the atomic multipole interaction energy using particle mesh Ewald summation and a neighbor list

EMPOLE1 Subroutine

"empole1" calculates the atomic multipole energy and first derivatives with respect to Cartesian coordinates

EMPOLE1A Subroutine

"empole1a" calculates the multipole energy and derivatives with respect to Cartesian coordinates using a pairwise double loop

EMPOLE1B Subroutine

"empole1b" calculates the multipole energy and derivatives with respect to Cartesian coordinates using a neighbor list

EMPOLE1C Subroutine

"empole1c" calculates the multipole energy and derivatives with respect to Cartesian coordinates using particle mesh Ewald summation and a double loop

EMPOLE1D Subroutine

"empole1d" calculates the multipole energy and derivatives with respect to Cartesian coordinates using particle mesh Ewald summation and a neighbor list

EMPOLE2 Subroutine

"empole2" calculates second derivatives of the multipole energy for a single atom at a time

EMPOLE2A Subroutine

"empole2a" computes multipole first derivatives for a single atom; used to get finite difference second derivatives

EMPOLE3 Subroutine

"empole3" calculates the electrostatic energy due to atomic multipole interactions, and partitions the energy among atoms

EMPOLE3A Subroutine

"empole3a" calculates the atomic multipole interaction energy using a double loop, and partitions the energy among atoms

EMPOLE3B Subroutine

"empole3b" calculates the atomic multipole interaction energy using a neighbor list, and partitions the energy among the atoms

EMPOLE3C Subroutine

"empole3c" calculates the atomic multipole interaction energy using a particle mesh Ewald summation and double loop, and partitions the energy among the atoms

EMPOLE3D Subroutine

"empole3d" calculates the atomic multipole interaction energy using particle mesh Ewald summation and a neighbor list, and partitions the energy among the atoms

EMREAL0C Subroutine

"emreal0c" evaluates the real space portion of the Ewald sum energy due to atomic multipoles using a double loop

EMREAL0D Subroutine

"emreal0d" evaluates the real space portion of the Ewald sum energy due to atomic multipoles using a neighbor list

EMREAL1C Subroutine

"emreal1c" evaluates the real space portion of the Ewald summation energy and gradient due to multipole interactions via a double loop

EMREAL1D Subroutine

"emreal1d" evaluates the real space portion of the Ewald summation energy and gradient due to multipole interactions via a neighbor list

EMREAL3C Subroutine

"emreal3c" evaluates the real space portion of the Ewald sum energy due to atomic multipole interactions and partitions the energy among the atoms

EMREAL3D Subroutine

"emreal3d" evaluates the real space portion of the Ewald sum energy due to atomic multipole interactions, and partitions the energy among the atoms using a pairwise neighbor list

EMRECIP Subroutine

"emrecip" evaluates the reciprocal space portion of the particle mesh Ewald energy due to atomic multipole interactions

EMRECIP1 Subroutine

"emrecip1" evaluates the reciprocal space portion of particle mesh Ewald summation energy and gradient due to multipoles

ENERGY Function

"energy" calls the subroutines to calculate the potential energy terms and sums up to form the total energy

ENP Subroutine

"enp" calculates the nonpolar implicit solvation energy as a sum of cavity and dispersion terms

ENP1 Subroutine

"enp1" calculates the nonpolar implicit solvation energy and derivatives as a sum of cavity and dispersion terms

ENP3 Subroutine

"enp3" calculates the nonpolar implicit solvation energy as a sum of cavity and dispersion terms; also partitions the energy among the atoms

ENRGYZE Subroutine

"enrgyze" is an auxiliary routine for the analyze program that performs the energy analysis and prints the total and intermolecular energies

EOPBEND Subroutine

"eopbend" computes the out-of-plane bend potential energy at trigonal centers via a Wilson-Decius-Cross or Allinger angle

EOPBEND1 Subroutine

"eopbend1" computes the out-of-plane bend potential energy and first derivatives at trigonal centers via a Wilson-Decius-Cross or Allinger angle

EOPBEND2 Subroutine

"eopbend2" calculates second derivatives of the out-of-plane bend energy via a Wilson-Decius-Cross or Allinger angle for a single atom using finite difference methods

EOPBEND2A Subroutine

"eopbend2a" calculates out-of-plane bend first derivatives at a trigonal center via a Wilson-Decius-Cross or Allinger angle; used in computation of finite difference second derivatives

EOPBEND3 Subroutine

"eopbend3" computes the out-of-plane bend potential energy at trigonal centers via a Wilson-Decius-Cross or Allinger angle; also partitions the energy among the atoms

EOPDIST Subroutine

"eopdist" computes the out-of-plane distance potential energy at trigonal centers via the central atom height

EOPDIST1 Subroutine

"eopdist1" computes the out-of-plane distance potential energy and first derivatives at trigonal centers via the central atom height

EOPDIST2 Subroutine

"eopdist2" calculates second derivatives of the out-of-plane distance energy for a single atom via the central atom height

EOPDIST3 Subroutine

"eopdist3" computes the out-of-plane distance potential energy at trigonal centers via the central atom height; also partitions the energy among the atoms

EPB Subroutine

"epb" calculates the implicit solvation energy via the Poisson-Boltzmann plus nonpolar implicit solvation

EPB1 Subroutine

"epb1" calculates the implicit solvation energy and derivatives via the Poisson-Boltzmann plus nonpolar implicit solvation

EPB1A Subroutine

"epb1a" calculates the solvation energy and gradients for the PB/NP solvation model

EPB3 Subroutine

"epb3" calculates the implicit solvation energy via the Poisson-Boltzmann model; also partitions the energy among the atoms

EPITORS Subroutine

"epitors" calculates the pi-system torsion potential energy

EPITORS1 Subroutine

"epitors1" calculates the pi-system torsion potential energy and first derivatives with respect to Cartesian coordinates

EPITORS2 Subroutine

"epitors2" calculates the second derivatives of the pi-system torsion energy for a single atom using finite difference methods

EPITORS2A Subroutine

"epitors2a" calculates the pi-system torsion first derivatives; used in computation of finite difference second derivatives

EPITORS3 Subroutine

"epitors3" calculates the pi-system torsion potential energy; also partitions the energy terms among the atoms

EPOLAR Subroutine

"epolar" calculates the polarization energy due to induced dipole interactions

EPOLAR0A Subroutine

"epolar0a" calculates the induced dipole polarization energy using a double loop, and partitions the energy among atoms

EPOLAR0B Subroutine

"epolar0b" calculates the induced dipole polarization energy using a neighbor list

EPOLAR0C Subroutine

"epolar0c" calculates the dipole polarization energy with respect to Cartesian coordinates using particle mesh Ewald summation and a double loop

EPOLAR0D Subroutine

"epolar0d" calculates the dipole polarization energy with respect to Cartesian coordinates using particle mesh Ewald summation and a neighbor list

EPOLAR0E Subroutine

"epolar0e" calculates the dipole polarizability interaction from the induced dipoles times the electric field

EPOLAR1 Subroutine

"epolar1" calculates the induced dipole polarization energy and first derivatives with respect to Cartesian coordinates

EPOLAR1A Subroutine

"epolar1a" calculates the dipole polarization energy and derivatives with respect to Cartesian coordinates using a pairwise double loop

EPOLAR1B Subroutine

"epolar1b" calculates the dipole polarization energy and derivatives with respect to Cartesian coordinates using a neighbor list

EPOLAR1C Subroutine

"epolar1c" calculates the dipole polarization energy and derivatives with respect to Cartesian coordinates using particle mesh Ewald summation and a double loop

EPOLAR1D Subroutine

"epolar1d" calculates the dipole polarization energy and derivatives with respect to Cartesian coordinates using particle mesh Ewald summation and a neighbor list

EPOLAR1E Subroutine

"epolar1e" calculates the dipole polarizability interaction from the induced dipoles times the electric field

EPOLAR2 Subroutine

"epolar2" calculates second derivatives of the dipole polarization energy for a single atom at a time

EPOLAR2A Subroutine

"epolar2a" computes polarization first derivatives for a single atom with respect to Cartesian coordinates; used to get finite difference second derivatives

EPOLAR3 Subroutine

"epolar3" calculates the induced dipole polarization energy, and partitions the energy among atoms

EPOLAR3A Subroutine

"epolar3a" calculates the induced dipole polarization energy using a double loop, and partitions the energy among atoms

EPOLAR3B Subroutine

"epolar3b" calculates the induced dipole polarization energy using a neighbor list, and partitions the energy among atoms

EPOLAR3C Subroutine

"epolar3c" calculates the polarization energy and analysis with respect to Cartesian coordinates using particle mesh Ewald and a double loop

EPOLAR3D Subroutine

"epolar3d" calculates the polarization energy and analysis with respect to Cartesian coordinates using particle mesh Ewald and a neighbor list

EPOLAR3E Subroutine

"epolar3e" calculates the dipole polarizability interaction from the induced dipoles times the electric field

EPREAL0C Subroutine

"epreal0c" calculates the induced dipole polarization energy using particle mesh Ewald summation and a double loop

EPREAL0D Subroutine

"epreal0d" calculates the induced dipole polarization energy using particle mesh Ewald summation and a neighbor list

EPREAL1C Subroutine

"epreal1c" evaluates the real space portion of the Ewald summation energy and gradient due to dipole polarization via a double loop

EPREAL1D Subroutine

"epreal1d" evaluates the real space portion of the Ewald summation energy and gradient due to dipole polarization via a neighbor list

EPREAL3C Subroutine

"epreal3c" calculates the induced dipole polarization energy and analysis using particle mesh Ewald summation and a double loop

EPREAL3D Subroutine

"epreal3d" calculates the induced dipole polarization energy and analysis using particle mesh Ewald and a neighbor list

EPRECIP Subroutine

"eprecip" evaluates the reciprocal space portion of particle mesh Ewald summation energy due to dipole polarization

EPRECIP1 Subroutine

"eprecip1" evaluates the reciprocal space portion of the particle mesh Ewald summation energy and gradient due to dipole polarization

EQUCLC Subroutine

EREPEL Subroutine

"erepel" calculates the Pauli exchange repulsion energy

EREPEL0A Subroutine

"erepel0a" calculates the Pauli repulsion interaction energy using a double loop

EREPEL0B Subroutine

"erepel0b" calculates the Pauli repulsion interaction energy using a pairwise neighbor list

EREPEL1 Subroutine

"erepel1" calculates the Pauli repulsion energy and first derivatives with respect to Cartesian coordinates

EREPEL1A Subroutine

"erepel1a" calculates the Pauli repulsion energy and first derivatives with respect to Cartesian coordinates using a pairwise double loop

EREPEL1B Subroutine

"erepel1b" calculates the Pauli repulsion energy and first derivatives with respect to Cartesian coordinates using a pariwise neighbor list

EREPEL2 Subroutine

"erepel2" calculates the second derivatives of the Pauli repulsion energy

EREPEL2A Subroutine

"erepel2a" computes Pauli repulsion first derivatives for a single atom via a double loop; used to get finite difference second derivatives

EREPEL3 Subroutine

"erepel3" calculates the Pauli repulsion energy and partitions the energy among the atoms

EREPEL3A Subroutine

"erepel3a" calculates the Pauli repulsion energy and also partitions the energy among the atoms using a double loop

EREPEL3B Subroutine

"erepel3b" calculates the Pauli repulsion energy and also partitions the energy among the atoms using a neighbor list

ERF Function

"erf" computes a numerical approximation to the value of the error function via a Chebyshev approximation

ERFC Function

"erfc" computes a numerical approximation to the value of the complementary error function via a Chebyshev approximation

ERFCORE Subroutine

"erfcore" evaluates erf(x) or erfc(x) for a real argument x; when called with mode set to 0 it returns erf, a mode of 1 returns erfc; uses rational functions that approximate erf(x) and erfc(x) to at least 18 significant decimal digits

ERFIK Subroutine

"erfik" compute the reaction field energy due to a single pair of atomic multipoles

ERFINV Function

"erfinv" evaluates the inverse of the error function for an argument in the range (-1,1) using a rational function approximation followed by cycles of Newton-Raphson correction

ERXNFLD Subroutine

"erxnfld" calculates the macroscopic reaction field energy arising from a set of atomic multipoles

ERXNFLD1 Subroutine

"erxnfld1" calculates the macroscopic reaction field energy and derivatives with respect to Cartesian coordinates

ERXNFLD2 Subroutine

"erxnfld2" calculates second derivatives of the macroscopic reaction field energy for a single atom at a time

ERXNFLD3 Subroutine

"erxnfld3" calculates the macroscopic reaction field energy, and also partitions the energy among the atoms

ESOLV Subroutine

"esolv" calculates the implicit solvation energy for surface area, generalized Born, generalized Kirkwood and Poisson-Boltzmann solvation models

ESOLV1 Subroutine

"esolv1" calculates the implicit solvation energy and first derivatives with respect to Cartesian coordinates for surface area, generalized Born, generalized Kirkwood and Poisson-Boltzmann solvation models

ESOLV2 Subroutine

"esolv2" calculates second derivatives of the implicit solvation energy for surface area, generalized Born, generalized Kirkwood and Poisson-Boltzmann solvation models

ESOLV2A Subroutine

"esolv2a" calculates second derivatives of the implicit solvation potential energy by finite differences

ESOLV2B Subroutine

"esolv2b" finds implicit solvation gradients needed for calculation of the Hessian matrix by finite differences

ESOLV3 Subroutine

"esolv3" calculates the implicit solvation energy for surface area, generalized Born, generalized Kirkwood and Poisson-Boltzmann solvation models; also partitions the energy among the atoms

ESTRBND Subroutine

"estrbnd" calculates the stretch-bend potential energy

ESTRBND1 Subroutine

"estrbnd1" calculates the stretch-bend potential energy and first derivatives with respect to Cartesian coordinates

ESTRBND2 Subroutine

"estrbnd2" calculates the stretch-bend potential energy second derivatives with respect to Cartesian coordinates

ESTRBND3 Subroutine

"estrbnd3" calculates the stretch-bend potential energy; also partitions the energy among the atoms

ESTRTOR Subroutine

"estrtor" calculates the stretch-torsion potential energy

ESTRTOR1 Subroutine

"estrtor1" calculates the stretch-torsion energy and first derivatives with respect to Cartesian coordinates

ESTRTOR2 Subroutine

"estrtor2" calculates the stretch-torsion potential energy second derivatives with respect to Cartesian coordinates

ESTRTOR3 Subroutine

"estrtor3" calculates the stretch-torsion potential energy; also partitions the energy terms among the atoms

ETORS Subroutine

"etors" calculates the torsional potential energy

ETORS0A Subroutine

"etors0a" calculates the torsional potential energy using a standard sum of Fourier terms

ETORS0B Subroutine

"etors0b" calculates the torsional potential energy for use with potential energy smoothing methods

ETORS1 Subroutine

"etors1" calculates the torsional potential energy and first derivatives with respect to Cartesian coordinates

ETORS1A Subroutine

"etors1a" calculates the torsional potential energy and first derivatives with respect to Cartesian coordinates using a standard sum of Fourier terms

ETORS1B Subroutine

"etors1b" calculates the torsional potential energy and first derivatives with respect to Cartesian coordinates for use with potential energy smoothing methods

ETORS2 Subroutine

"etors2" calculates the second derivatives of the torsional energy for a single atom

ETORS2A Subroutine

"etors2a" calculates the second derivatives of the torsional energy for a single atom using a standard sum of Fourier terms

ETORS2B Subroutine

"etors2b" calculates the second derivatives of the torsional energy for a single atom for use with potential energy smoothing methods

ETORS3 Subroutine

"etors3" calculates the torsional potential energy; also partitions the energy among the atoms

ETORS3A Subroutine

"etors3a" calculates the torsional potential energy using a standard sum of Fourier terms and partitions the energy among the atoms

ETORS3B Subroutine

"etors3b" calculates the torsional potential energy for use with potential energy smoothing methods and partitions the energy among the atoms

ETORTOR Subroutine

"etortor" calculates the torsion-torsion potential energy

ETORTOR1 Subroutine

"etortor1" calculates the torsion-torsion energy and first derivatives with respect to Cartesian coordinates

ETORTOR2 Subroutine

"etortor2" calculates the torsion-torsion potential energy second derivatives with respect to Cartesian coordinates

ETORTOR3 Subroutine

"etortor3" calculates the torsion-torsion potential energy; also partitions the energy terms among the atoms

EUREY Subroutine

"eurey" calculates the Urey-Bradley 1-3 interaction energy

EUREY1 Subroutine

"eurey1" calculates the Urey-Bradley interaction energy and its first derivatives with respect to Cartesian coordinates

EUREY2 Subroutine

"eurey2" calculates second derivatives of the Urey-Bradley interaction energy for a single atom at a time

EUREY3 Subroutine

"eurey3" calculates the Urey-Bradley energy; also partitions the energy among the atoms

EVCORR Subroutine

"evcorr" computes the long range van der Waals correction to the energy via numerical integration

EVCORR1 Subroutine

"evcorr1" computes the long range van der Waals correction to the energy and virial via numerical integration

EWALDCOF Subroutine

"ewaldcof" finds an Ewald coefficient such that all terms beyond the specified cutoff distance will have a value less than a specified tolerance

EWCA Subroutine

"ewca" find the Weeks-Chandler-Andersen dispersion energy of a solute using an HCT-like method

EWCA1 Subroutine

"ewca1" finds the Weeks-Chandler-Anderson dispersion energy and derivatives of a solute

EWCA3 Subroutine

"ewca3" find the Weeks-Chandler-Andersen dispersion energy of a solute; also partitions the energy among the atoms

EWCA3X Subroutine

"ewca3x" finds the Weeks-Chandler-Anderson dispersion energy of a solute using a numerical "onion shell" method; also partitions the energy among the atoms

EWCAX Subroutine

"ewcax" finds the Weeks-Chandler-Anderson dispersion energy of a solute using a numerical "onion shell" method

EXPLORE Subroutine

"explore" uses simulated annealing on an initial crude embedded distance geoemtry structure to refine versus the bound, chirality, planarity and torsional error functions

EXTENT Subroutine

"extent" finds the largest interatomic distance in a system

EXTRA Subroutine

"extra" calculates any additional user defined potential energy contribution

EXTRA1 Subroutine

"extra1" calculates any additional user defined potential energy contribution and its first derivatives

EXTRA2 Subroutine

"extra2" calculates second derivatives of any additional user defined potential energy contribution for a single atom at a time

EXTRA3 Subroutine

"extra3" calculates any additional user defined potential contribution and also partitions the energy among the atoms

FATAL Subroutine

"fatal" terminates execution due to a user request, a severe error or some other nonstandard condition

FFTBACK Subroutine

"fftback" performs a 3-D FFT backward transform via a single 3-D transform or three separate 1-D transforms

FFTCLOSE Subroutine

"fftclose" does cleanup after performing a 3-D FFT by destroying the FFTW plans for the forward and backward transforms

FFTFRONT Subroutine

"fftfront" performs a 3-D FFT forward transform via a single 3-D transform or three separate 1-D transforms

FFTSETUP Subroutine

"fftsetup" does initialization for a 3-D FFT to be computed via either the FFTPACK or FFTW libraries

FIELD Subroutine

"field" sets the force field potential energy functions from a parameter file and modifications specified in a keyfile

FINAL Subroutine

"final" performs any final program actions such as deallocation of global memory, prints a status message, and then pauses if necessary to avoid closing the execution window

FINDATM Subroutine

"findatm" locates a specific PDB atom name type within a range of atoms from the PDB file, returns zero if the name type was not found

FITRSD Subroutine

"fitrsd" computes residuals for electrostatic potential fitting including total charge restraints, dipole and quadrupole moment targets, and restraints to initial parameter values

FITTORS Subroutine

"fittors" refines torsion parameters based on a quantum mechanical optimized energy surface

FIXFRAME Subroutine

"fixframe" is a service routine that alters the local frame definition for specified atoms

FIXPDB Subroutine

"fixpdb" corrects problems with PDB files by converting residue and atom names to the standard forms used by Tinker

FIXPOLE Subroutine

"fixpole" performs unit conversion of the multipole components, rounds moments to desired precision, and enforces integer net charge and traceless quadrupoles

FLATTEN Subroutine

"flatten" sets the type of smoothing method and the extent of surface deformation for use with potential energy smoothing

FPHI_MPOLE Subroutine

"fphi_mpole" extracts the permanent multipole potential from the particle mesh Ewald grid

FPHI_TO_CPHI Subroutine

"fphi_to_cphi" transforms the reciprocal space potential from fractional to Cartesian coordinates

FPHI_UIND Subroutine

"fphi_uind" extracts the induced dipole potential from the particle mesh Ewald grid

FRACDIST Subroutine

"fracdist" computes a normalized distribution of the pairwise fractional distances between the smoothed upper and lower bounds

FRAC_TO_CART Subroutine

"frac_to_cart" computes a transformation matrix to convert a multipole object in fraction coordinates to Cartesian

FRAME13 Subroutine

"frame13" finds local coordinate frame defining atoms in cases where the use of 1-3 connected atoms is required

FREEUNIT Function

"freeunit" finds an unopened Fortran I/O unit and returns its numerical value from 1 to 99; the units already assigned to "input" and "iout" (usually 5 and 6) are skipped since they have special meaning as the default I/O units

GAMMLN Function

"gammln" uses a series expansion due to Lanczos to compute the natural logarithm of the Gamma function at "x" in [0,1]

GAUSSJORDAN Subroutine

"gaussjordan" solves a system of linear equations by using the method of Gaussian elimination with partial pivoting

GDA Program

"gda" implements Gaussian Density Annealing (GDA) algorithm for global optimization via simulated annealing

GDA1 Subroutine

GDA2 Function

GDA3 Subroutine

GDASTAT Subroutine

for a GDA integration step; also saves the coordinates

GENDOT Subroutine

"gendot" finds the coordinates of a specified number of surface points for a sphere with the input radius and coordinate center

GEODESIC Subroutine

"geodesic" smooths the upper and lower distance bounds via the triangle inequality using a sparse matrix version of a shortest path algorithm

GEOMETRY Function

"geometry" finds the value of the interatomic distance, angle or dihedral angle defined by two to four input atoms

GETARC Subroutine

"getarc" asks for a coordinate archive or trajectory file name, then reads in the initial set of coordinates

GETBASE Subroutine

"getbase" finds the base heavy atoms for a single nucleotide residue and copies the names and coordinates to the Protein Data Bank file

GETCHUNK Subroutine

"getchunk" determines the number of grid point "chunks" used along each axis of the PME grid for parallelization

GETINT Subroutine

"getint" asks for an internal coordinate file name, then reads the internal coordinates and computes Cartesian coordinates

GETKEY Subroutine

"getkey" finds a valid keyfile and stores its contents as line images for subsequent keyword parameter searching

GETMOL Subroutine

"getmol" asks for a MDL MOL molecule file name, then reads the coordinates from the file

GETMOL2 Subroutine

"getmol2" asks for a Tripos MOL2 molecule file name, then reads the coordinates from the file

GETMONITOR Subroutine

GETNUCH Subroutine

"getnuch" finds the nucleotide hydrogen atoms for a single residue and copies the names and coordinates to the Protein Data Bank file

GETNUMB Subroutine

"getnumb" searches an input string from left to right for an integer and puts the numeric value in "number"; returns zero with "next" unchanged if no integer value is found

GETPDB Subroutine

"getpdb" asks for a Protein Data Bank file name, then reads in the coordinates file

GETPRB Subroutine

"getprb" tests for a possible probe position at the interface between three neighboring atoms

GETPRM Subroutine

"getprm" finds the potential energy parameter file and then opens and reads the parameters

GETPROH Subroutine

"getproh" finds the hydrogen atoms for a single amino acid residue and copies the names and coordinates to the Protein Data Bank file

GETREF Subroutine

"getref" copies structure information from the reference area into the standard variables for the current system structure

GETSEQ Subroutine

"getseq" asks the user for the amino acid sequence and torsional angle values needed to define a peptide

GETSEQN Subroutine

"getseqn" asks the user for the nucleotide sequence and torsional angle values needed to define a nucleic acid

GETSIDE Subroutine

"getside" finds the side chain heavy atoms for a single amino acid residue and copies the names and coordinates to the Protein Data Bank file

GETSTRING Subroutine

"getstring" searches for a quoted text string within an input character string; the region between the first and second double quote is returned as the "text"; if the actual text is too long, only the first part is returned

GETTEXT Subroutine

"gettext" searches an input string for the first string of non-blank characters; the region from a non-blank character to the first space or tab is returned as "text"; if the actual text is too long, only the first part is returned

GETTIME Subroutine

"gettime" finds the elapsed wall clock and CPU times in seconds since the last call to "settime"

GETTOR Subroutine

"gettor" tests for a possible torus position at the interface between two atoms, and finds the torus radius, center and axis

GETWORD Subroutine

"getword" searches an input string for the first alphabetic character (A-Z or a-z); the region from this first character to the first blank space or separator is returned as a "word"; if the actual word is too long, only the first part is returned

GETXYZ Subroutine

"getxyz" asks for a Cartesian coordinate file name, then reads in the coordinates file

GHMCSTEP Subroutine

"ghmcstep" performs a single stochastic dynamics time step via the generalized hybrid Monte Carlo (GHMC) algorithm to ensure exact sampling from the Boltzmann density

GHMCTERM Subroutine

"ghmcterm" finds the friction and fluctuation terms needed to update velocities during GHMC stochastic dynamics

GRADFAST Subroutine

"gradfast" calculates the potential energy and first derivatives for the fast-evolving local valence potential energy terms

GRADIENT Subroutine

"gradient" calls subroutines to calculate the potential energy and first derivatives with respect to Cartesian coordinates

GRADRGD Subroutine

"gradrgd" calls subroutines to calculate the potential energy and first derivatives with respect to rigid body coordinates

GRADROT Subroutine

"gradrot" calls subroutines to calculate the potential energy and its torsional first derivatives

GRADSLOW Subroutine

"gradslow" calculates the potential energy and first derivatives for the slow-evolving nonbonded potential energy terms

GRAFIC Subroutine

"grafic" outputs the upper & lower triangles and diagonal of a square matrix in a schematic form for visual inspection

GRID_DISP Subroutine

"grid_disp" places the damped dispersion coefficients onto the particle mesh Ewald grid

GRID_MPOLE Subroutine

"grid_mpole" places the fractional atomic multipoles onto the particle mesh Ewald grid

GRID_PCHG Subroutine

"grid_pchg" places the fractional atomic partial charges onto the particle mesh Ewald grid

GRID_UIND Subroutine

"grid_uind" places the fractional induced dipoles onto the particle mesh Ewald grid

GROUPS Subroutine

"groups" tests a set of atoms to see if all are members of a single atom group or a pair of atom groups; if so, then the correct intra- or intergroup weight is assigned

GRPLINE Subroutine

"grpline" tests each atom group for linearity of the sites contained in the group

GSORT Subroutine

"gsort" uses the Gram-Schmidt algorithm to build orthogonal vectors for sliding block interative matrix diagonalization

GYRATE Subroutine

"gyrate" computes the radius of gyration of a molecular system from its atomic coordinates; only active atoms are included

HANGLE Subroutine

"hangle" constructs hybrid angle bending parameters given an initial state, final state and "lambda" value

HATOM Subroutine

"hatom" assigns a new atom type to each hybrid site

HBOND Subroutine

"hbond" constructs hybrid bond stretch parameters given an initial state, final state and "lambda" value

HCHARGE Subroutine

"hcharge" constructs hybrid charge interaction parameters given an initial state, final state and "lambda" value

HDIPOLE Subroutine

"hdipole" constructs hybrid dipole interaction parameters given an initial state, final state and "lambda" value

HESSBLK Subroutine

"hessblk" calls subroutines to calculate the Hessian elements for each atom in turn with respect to Cartesian coordinates

HESSIAN Subroutine

"hessian" calls subroutines to calculate the Hessian elements for each atom in turn with respect to Cartesian coordinates

HESSRGD Subroutine

"hessrgd" computes the numerical Hessian elements with respect to rigid body coordinates via 6*ngroup+1 gradient evaluations

HESSROT Subroutine

"hessrot" computes numerical Hessian elements with respect to torsional angles; either the diagonal or the full matrix can be calculated; the full matrix needs nomega+1 gradient evaluations while the diagonal needs just two evaluations

HETATOM Subroutine

"hetatom" translates water molecules and ions in Protein Data Bank format to a Cartesian coordinate file and sequence file

HIMPTOR Subroutine

"himptor" constructs hybrid improper torsional parameters given an initial state, final state and "lambda" value

HOOVER Subroutine

"hoover" applies a combined thermostat and barostat via a Nose-Hoover chain algorithm

HSTRBND Subroutine

"hstrbnd" constructs hybrid stretch-bend parameters given an initial state, final state and "lambda" value

HSTRTOR Subroutine

"hstrtor" constructs hybrid stretch-torsion parameters given an initial state, final state and "lambda" value

HTORS Subroutine

"htors" constructs hybrid torsional parameters for a given initial state, final state and "lambda" value

HVDW Subroutine

"hvdw" constructs hybrid van der Waals parameters given an initial state, final state and "lambda" value

HYBRID Subroutine

"hybrid" constructs the hybrid hamiltonian for a specified initial state, final state and mutation parameter "lambda"

IJKPTS Subroutine

"ijkpts" stores a set of indices used during calculation of macroscopic reaction field energetics

IMAGE Subroutine

"image" takes the components of pairwise distance between two points in a periodic box and converts to the components of the minimum image distance

IMAGEN Subroutine

"imagen" takes the components of pairwise distance between two points and converts to the components of the minimum image distance

IMAGER Subroutine

"imager" takes the components of pairwise distance between two points in the same or neighboring periodic boxes and converts to the components of the minimum image distance

IMPOSE Subroutine

"impose" performs the least squares best superposition of two atomic coordinate sets via a quaternion method; upon return, the first coordinate set is unchanged while the second set is translated and rotated to give best fit; the final root mean square fit is returned in "rmsvalue"

INDTCGA Subroutine

"indtcga" computes the induced dipoles and intermediates used in polarization force calculation for the TCG method with dp cross terms = true, initial guess mu0 = 0 and using a diagonal preconditioner

INDTCGB Subroutine

"indtcgb" computes the induced dipoles and intermediates used in polarization force calculation for the TCG method with dp cross terms = true, initial guess mu0 = direct and using diagonal preconditioner

INDUCE Subroutine

"induce" computes the induced dipole moments at polarizable sites due to direct or mutual polarization

INDUCE0A Subroutine

"induce0a" computes the induced dipole moments at polarizable sites using a preconditioned conjugate gradient solver

INDUCE0B Subroutine

"induce0b" computes and stores the induced dipoles via the truncated conjugate gradient (TCG) method

INDUCE0C Subroutine

"induce0c" computes the induced dipole moments at polarizable sites for generalized Kirkwood SCRF and vacuum environments

INDUCE0D Subroutine

"induce0d" computes the induced dipole moments at polarizable sites for Poisson-Boltzmann SCRF and vacuum environments

INEDGE Subroutine

"inedge" inserts a concave edge into the linked list for its temporary torus

INERTIA Subroutine

"inertia" computes the principal moments of inertia for the system, and optionally translates the center of mass to the origin and rotates the principal axes onto the global axes

INITATOM Subroutine

"initatom" sets the atomic symbol, standard atomic weight, van der Waals radius and covalent radius for each element in the periodic table

INITERR Function

"initerr" is the initial error function and derivatives for a distance geometry embedding; it includes components from the local geometry and torsional restraint errors

INITIAL Subroutine

"initial" sets up original values for some parameters and variables that might not otherwise get initialized

INITMMFF Subroutine

"initmmff" initializes some parameter values for the Merck Molecular force field

INITPRM Subroutine

"initprm" completely initializes a force field by setting all parameters to zero and using defaults for control values

INITRES Subroutine

"initres" sets biopolymer residue names and biotype codes used in PDB file conversion and automated generation of structures

INITROT Subroutine

"initrot" sets the torsional angles which are to be rotated in subsequent computation, by default automatically selects all rotatable single bonds; optionally makes atoms inactive when they are not moved by any torsional rotation

INSERT Subroutine

"insert" adds the specified atom to the Cartesian coordinates list and shifts the remaining atoms

INTEDIT Program

"intedit" allows the user to extract information from or alter the values within an internal coordinates file

INTERPOL Subroutine

"interpol" computes intergroup induced dipole moments for use during removal of intergroup polarization

INTXYZ Program

"intxyz" takes as input an internal coordinates file, converts to and then writes out Cartesian coordinates

INVBETA Function

"invbeta" computes the inverse Beta distribution function via a combination of Newton iteration and bisection search

INVERT Subroutine

"invert" inverts a matrix using the Gauss-Jordan method

IPEDGE Subroutine

"ipedge" inserts convex edge into linked list for atom

JACOBI Subroutine

"jacobi" performs a matrix diagonalization of a real symmetric matrix by the method of Jacobi rotations

JUSTIFY Subroutine

"justify" converts a text string to right justified format with leading blank spaces

KANGANG Subroutine

"kangang" assigns the parameters for angle-angle cross term interactions and processes new or changed parameter values

KANGLE Subroutine

"kangle" assigns the force constants and ideal angles for the bond angles; also processes new or changed parameters

KANGLEM Subroutine

"kanglem" assigns the force constants and ideal angles for bond angles according to the Merck Molecular Force Field (MMFF)

KANGTOR Subroutine

"kangtor" assigns parameters for angle-torsion interactions and processes new or changed parameter values

KATOM Subroutine

"katom" assigns an atom type definitions to each atom in the structure and processes any new or changed values

KBOND Subroutine

"kbond" assigns a force constant and ideal bond length to each bond in the structure and processes any new or changed parameter values

KBONDM Subroutine

"kbondm" assigns a force constant and ideal bond length to each bond according to the Merck Molecular Force Field (MMFF)

KCHARGE Subroutine

"kcharge" assigns partial charges to the atoms within the structure and processes any new or changed values

KCHARGEM Subroutine

"kchargem" assigns partial charges to the atoms according to the Merck Molecular Force Field (MMFF)

KCHGTRN Subroutine

"kchgtrn" assigns charge magnitude and damping parameters for charge transfer interactions and processes any new or changed values for these parameters

KCHIRAL Subroutine

"kchiral" determines the target value for each chirality and planarity restraint as the signed volume of the parallelpiped spanned by vectors from a common atom to each of three other atoms

KDIPOLE Subroutine

"kdipole" assigns bond dipoles to the bonds within the structure and processes any new or changed values

KDISP Subroutine

"kdisp" assigns C6 coefficients and damping parameters for dispersion interactions and processes any new or changed values for these parameters

KENEG Subroutine

"keneg" applies primary and secondary electronegativity bond length corrections to applicable bond parameters

KEWALD Subroutine

"kewald" assigns particle mesh Ewald parameters and options for a periodic system

KEXTRA Subroutine

"kextra" assigns parameters to any additional user defined potential energy contribution

KGB Subroutine

"kgb" initializes parameters needed for the generalized Born implicit solvation models

KGEOM Subroutine

"kgeom" asisgns parameters for geometric restraint terms to be included in the potential energy calculation

KGK Subroutine

"kgk" initializes parameters needed for the generalized Kirkwood implicit solvation model

KHPMF Subroutine

"khpmf" initializes parameters needed for the hydrophobic potential of mean force nonpolar implicit solvation model

KIMPROP Subroutine

"kimprop" assigns potential parameters to each improper dihedral in the structure and processes any changed values

KIMPTOR Subroutine

"kimptor" assigns torsional parameters to each improper torsion in the structure and processes any changed values

KINAUX Subroutine

"kinaux" computes the total kinetic energy and temperature for auxiliary dipole variables used in iEL polarization

KINETIC Subroutine

"kinetic" computes the total kinetic energy and kinetic energy contributions to the pressure tensor by summing over velocities

KMETAL Subroutine

"kmetal" assigns ligand field parameters to transition metal atoms and processes any new or changed parameter values

KMPOLE Subroutine

"kmpole" assigns atomic multipole moments to the atoms of the structure and processes any new or changed values

KNP Subroutine

"knp" initializes parameters needed for the cavity-plus- dispersion nonpolar implicit solvation model

KONVEC Subroutine

"konvec" finds a Hessian-vector product via finite-difference evaluation of the gradient based on atomic displacements

KOPBEND Subroutine

"kopbend" assigns the force constants for out-of-plane bends at trigonal centers via Wilson-Decius-Cross or Allinger angles; also processes any new or changed parameter values

KOPBENDM Subroutine

"kopbendm" assigns the force constants for out-of-plane bends according to the Merck Molecular Force Field (MMFF)

KOPDIST Subroutine

"kopdist" assigns the force constants for out-of-plane distance at trigonal centers via the central atom height; also processes any new or changed parameter values

KORBIT Subroutine

"korbit" assigns pi-orbital parameters to conjugated systems and processes any new or changed parameters

KPB Subroutine

"kpb" assigns parameters needed for the Poisson-Boltzmann implicit solvation model implemented via APBS

KPITORS Subroutine

"kpitors" assigns pi-system torsion parameters to torsions needing them, and processes any new or changed values

KPOLAR Subroutine

"kpolar" assigns atomic dipole polarizabilities to the atoms within the structure and processes any new or changed values

KREPEL Subroutine

"krepel" assigns the size values, exponential parameter and number of valence electrons for Pauli repulsion interactions and processes any new or changed values for these parameters

KSA Subroutine

"ksa" initializes parameters needed for surface area-based implicit solvation models including ASP and SASA

KSOLV Subroutine

"ksolv" assigns implicit solvation energy parameters for the surface area, generalized Born, generalized Kirkwood, Poisson-Boltzmann, cavity-dispersion and HPMF models

KSTRBND Subroutine

"kstrbnd" assigns parameters for stretch-bend interactions and processes new or changed parameter values

KSTRBNDM Subroutine

"kstrbndm" assigns parameters for stretch-bend interactions according to the Merck Molecular Force Field (MMFF)

KSTRTOR Subroutine

"kstrtor" assigns stretch-torsion parameters to torsions needing them, and processes any new or changed values

KTORS Subroutine

"ktors" assigns torsional parameters to each torsion in the structure and processes any new or changed values

KTORSM Subroutine

"ktorsm" assigns torsional parameters to each torsion according to the Merck Molecular Force Field (MMFF)

KTORTOR Subroutine

"ktortor" assigns torsion-torsion parameters to adjacent torsion pairs and processes any new or changed values

KUREY Subroutine

"kurey" assigns the force constants and ideal distances for the Urey-Bradley 1-3 interactions; also processes any new or changed parameter values

KVDW Subroutine

"kvdw" assigns the parameters to be used in computing the van der Waals interactions and processes any new or changed values for these parameters

LATTICE Subroutine

"lattice" stores the periodic box dimensions and sets angle values to be used in computing fractional coordinates

LBFGS Subroutine

"lbfgs" is a limited memory BFGS quasi-newton nonlinear optimization routine

LIGASE Subroutine

"ligase" translates a nucleic acid structure in Protein Data Bank format to a Cartesian coordinate file and sequence file

LIGHTS Subroutine

"lights" computes the set of nearest neighbor interactions using the method of lights algorithm

LINBODY Subroutine

"linbody" finds the angular velocity of a linear rigid body given the inertia tensor and angular momentum

LMSTEP Subroutine

"lmstep" computes a Levenberg-Marquardt step during a nonlinear least squares calculation using ideas from the MINPACK LMPAR routine and the internal doubling strategy of Dennis and Schnabel

LOCALMIN Subroutine

"localmin" is used during normal mode local search to perform a Cartesian coordinate energy minimization

LOCALRGD Subroutine

"localrgd" is used during the PSS local search procedure to perform a rigid body energy minimization

LOCALROT Subroutine

"localrot" is used during the PSS local search procedure to perform a torsional space energy minimization

LOCALXYZ Subroutine

"localxyz" is used during the potential smoothing and search procedure to perform a local optimization at the current smoothing level

LOCERR Function

"locerr" is the local geometry error function and derivatives including the 1-2, 1-3 and 1-4 distance bound restraints

LOWCASE Subroutine

"lowcase" converts a text string to all lower case letters

MAJORIZE Subroutine

"majorize" refines the projected coordinates by attempting to minimize the least square residual between the trial distance matrix and the distances computed from the coordinates

MAKEBAR Subroutine

MAKEBOX Subroutine

"makebox" builds a periodic box of a desired size by randomly copying a specified number of monomers into a target box size, followed by optional excluded volume refinement

MAKEINT Subroutine

"makeint" converts Cartesian to internal coordinates where selection of internal coordinates is controlled by "mode"

MAKEPDB Subroutine

"makepdb" cconstructs a Protein Data Bank file from a set of Cartesian coordinates with special handling for systems consisting of biopolymer chains, ligands and water molecules

MAKEREF Subroutine

"makeref" copies the information contained in the "xyz" file of the current structure into corresponding reference areas

MAKEXYZ Subroutine

"makexyz" generates a complete set of Cartesian coordinates for a full structure from the internal coordinate values

MAPCHECK Subroutine

"mapcheck" checks the current minimum energy structure for possible addition to the master list of local minima

MATCH1 Subroutine

"match1" finds and stores the first multipole component found on a line of output from Stone's GDMA program

MATCH2 Subroutine

"match2" finds and stores the second multipole component found on a line of output from Stone's GDMA program

MATCH3 Subroutine

"match3" finds and stores the third multipole component found on a line of output from Stone's GDMA program

MAXWELL Function

"maxwell" returns a speed in Angstroms/picosecond randomly selected from a 3-D Maxwell-Boltzmann distribution for the specified particle mass and system temperature

MBUILD Subroutine

"mbuild" performs a complete rebuild of the atomic multipole electrostatic neighbor list for all sites

MCM1 Function

"mcm1" is a service routine that computes the energy and gradient for truncated Newton optimization in Cartesian coordinate space

MCM2 Subroutine

"mcm2" is a service routine that computes the sparse matrix Hessian elements for truncated Newton optimization in Cartesian coordinate space

MCMSTEP Function

"mcmstep" implements the minimization phase of an MCM step via Cartesian minimization following a Monte Carlo step

MDINIT Subroutine

"mdinit" initializes the velocities and accelerations for a molecular dynamics trajectory, including restarts

MDREST Subroutine

"mdrest" finds and removes any translational or rotational kinetic energy of the overall system center of mass

MDSAVE Subroutine

"mdsave" writes molecular dynamics trajectory snapshots and auxiliary files with velocity, force or induced dipole data; also checks for user requested termination of a simulation

MDSTAT Subroutine

"mdstat" is called at each molecular dynamics time step to form statistics on various average values and fluctuations, and to periodically save the state of the trajectory

MEASFN Subroutine

MEASFQ Subroutine

MEASFS Subroutine

MEASPM Subroutine

"measpm" computes the volume of a single prism section of the full interior polyhedron

MECHANIC Subroutine

"mechanic" sets up needed parameters for the potential energy calculation and reads in many of the user selectable options

MERGE Subroutine

"merge" combines the reference and current structures into a single new "current" structure containing the reference atoms followed by the atoms of the current structure

METRIC Subroutine

"metric" takes as input the trial distance matrix and computes the metric matrix of all possible dot products between the atomic vectors and the center of mass using the law of cosines and the following formula for the distances to the center of mass:

MIDERR Function

"miderr" is the secondary error function and derivatives for a distance geometry embedding; it includes components from the distance bounds, local geometry, chirality and torsional restraint errors

MINIMIZ1 Function

"minimiz1" is a service routine that computes the energy and gradient for a low storage BFGS optimization in Cartesian coordinate space

MINIMIZE Program

"minimize" performs energy minimization in Cartesian coordinate space using a low storage BFGS nonlinear optimization

MINIROT Program

"minirot" performs an energy minimization in torsional angle space using a low storage BFGS nonlinear optimization

MINIROT1 Function

"minirot1" is a service routine that computes the energy and gradient for a low storage BFGS nonlinear optimization in torsional angle space

MINPATH Subroutine

"minpath" is a routine for finding the triangle smoothed upper and lower bounds of each atom to a specified root atom using a sparse variant of the Bellman-Ford shortest path algorithm

MINRIGID Program

"minrigid" performs an energy minimization of rigid body atom groups using a low storage BFGS nonlinear optimization

MINRIGID1 Function

"minrigid1" is a service routine that computes the energy and gradient for a low storage BFGS nonlinear optimization of rigid bodies

MLIGHT Subroutine

"mlight" performs a complete rebuild of the atomic multipole pair neighbor list for all sites using the method of lights

MLIST Subroutine

"mlist" performs an update or a complete rebuild of the nonbonded neighbor lists for atomic multipoles

MMID Subroutine

"mmid" implements a modified midpoint method to advance the integration of a set of first order differential equations

MODECART Subroutine

MODERGD Subroutine

MODEROT Subroutine

MODESRCH Subroutine

MODETORS Subroutine

MODULI Subroutine

"moduli" sets the moduli of the inverse discrete Fourier transform of the B-splines

MOL2XYZ Program

"mol2xyz" takes as input a Tripos MOL2 coordinates file, converts to and then writes out Cartesian coordinates

MOLECULE Subroutine

"molecule" counts the molecules, assigns each atom to its molecule and computes the mass of each molecule

MOLMERGE Subroutine

"molmerge" connects fragments and removes duplicate atoms during generation of a unit cell from an asymmetric unit

MOLSETUP Subroutine

"molsetup" generates trial parameters needed to perform polarizable multipole calculations on a structure read from distributed multipole analysis output

MOLUIND Subroutine

"moluind" computes the molecular induced dipole components in the presence of an external electric field

MOLXYZ Program

"molxyz" takes as input a MDL MOL coordinates file, converts to and then writes out Cartesian coordinates

MOMENTS Subroutine

"moments" computes the total electric charge, dipole and quadrupole moments for the active atoms as a sum over the partial charges, bond dipoles and atomic multipole moments

MOMFULL Subroutine

"momfull" computes the electric moments for the full system as a sum over the partial charges, bond dipoles and atomic multipole moments

MOMYZE Subroutine

"momyze" finds and prints the total charge, dipole moment components, radius of gyration and moments of inertia

MONTE Program

"monte" performs a Monte Carlo-Minimization conformational search using Cartesian single atom or torsional move sets

MUTATE Subroutine

"mutate" constructs the hybrid hamiltonian for a specified initial state, final state and mutation parameter "lambda"

NBLIST Subroutine

"nblist" builds and maintains nonbonded pair neighbor lists for vdw, dispersion, electrostatic and polarization terms

NEARBY Subroutine

"nearby" finds all of the through-space neighbors of each atom for use in surface area and volume calculations

NEEDUPDATE Subroutine

NEWATM Subroutine

"newatm" creates and defines an atom needed for the Cartesian coordinates file, but which may not present in the original Protein Data Bank file

NEWTON Program

"newton" performs an energy minimization in Cartesian coordinate space using a truncated Newton method

NEWTON1 Function

"newton1" is a service routine that computes the energy and gradient for truncated Newton optimization in Cartesian coordinate space

NEWTON2 Subroutine

"newton2" is a service routine that computes the sparse matrix Hessian elements for truncated Newton optimization in Cartesian coordinate space

NEWTROT Program

"newtrot" performs an energy minimization in torsional angle space using a truncated Newton conjugate gradient method

NEWTROT1 Function

"newtrot1" is a service routine that computes the energy and gradient for truncated Newton conjugate gradient optimization in torsional angle space

NEWTROT2 Subroutine

"newtrot2" is a service routine that computes the sparse matrix Hessian elements for truncated Newton optimization in torsional angle space

NEXTARG Subroutine

"nextarg" finds the next unused command line argument and returns it in the input character string

NEXTTEXT Function

"nexttext" finds and returns the location of the first non-blank character within an input text string; zero is returned if no such character is found

NORMAL Function

"normal" generates a random number from a normal Gaussian distribution with a mean of zero and a variance of one

NOSE Subroutine

"nose" performs a single molecular dynamics time step via a Nose-Hoover extended system isothermal-isobaric algorithm

NSPLINE Subroutine

"nspline" computes coefficients for an nonperiodic cubic spline with natural boundary conditions where the first and last second derivatives are already known

NUCBASE Subroutine

"nucbase" builds the side chain for a single nucleotide base in terms of internal coordinates

NUCCHAIN Subroutine

"nucchain" builds up the internal coordinates for a nucleic acid sequence from the sugar type, backbone and glycosidic torsional values

NUCLEIC Program

"nucleic" builds the internal and Cartesian coordinates of a polynucleotide from nucleic acid sequence and torsional angle values for the nucleic acid backbone and side chains

NUMBER Function

"number" converts a text numeral into an integer value; the input string must contain only numeric characters

NUMERAL Subroutine

"numeral" converts an input integer number into the corresponding right- or left-justified text numeral

NUMGRAD Subroutine

"numgrad" computes the gradient of the objective function "fvalue" with respect to Cartesian coordinates of the atoms via a one-sided or two-sided numerical differentiation

OCVM Subroutine

"ocvm" is an optimally conditioned variable metric nonlinear optimization routine without line searches

OLDATM Subroutine

"oldatm" get the Cartesian coordinates for an atom from the Protein Data Bank file, then assigns the atom type and atomic connectivities

OPBGUESS Function

"opbguess" sets approximate out-of-plane bend force constants based on atom type and connected atoms

OPENEND Subroutine

"openend" opens a file on a Fortran unit such that the position is set to the bottom for appending to the end of the file

OPREP Subroutine

"oprep" sets up the frictional and random terms needed to update positions and velocities for the BAOAB integrator

OPTFIT Function

OPTIMIZ1 Function

"optimiz1" is a service routine that computes the energy and gradient for optimally conditioned variable metric optimization in Cartesian coordinate space

OPTIMIZE Program

"optimize" performs energy minimization in Cartesian coordinate space using an optimally conditioned variable metric method

OPTINIT Subroutine

"optinit" initializes values and keywords used by multiple structure optimization methods

OPTIROT Program

"optirot" performs an energy minimization in torsional angle space using an optimally conditioned variable metric method

OPTIROT1 Function

"optirot1" is a service routine that computes the energy and gradient for optimally conditioned variable metric optimization in torsional angle space

OPTRIGID Program

"optrigid" performs an energy minimization of rigid body atom groups using an optimally conditioned variable metric method

OPTRIGID1 Function

"optrigid1" is a service routine that computes the energy and gradient for optimally conditioned variable metric optimization of rigid bodies

OPTSAVE Subroutine

"optsave" is used by the optimizers to write imtermediate coordinates and other relevant information; also checks for user requested termination of an optimization

ORBITAL Subroutine

"orbital" finds and organizes lists of atoms in a pisystem, bonds connecting pisystem atoms and torsions whose central atoms are both pisystem atoms

ORIENT Subroutine

"orient" computes a set of reference Cartesian coordinates in standard orientation for each rigid body atom group

ORTHOG Subroutine

"orthog" performs an orthogonalization of an input matrix via the modified Gram-Schmidt algorithm

OVERLAP Subroutine

"overlap" computes the overlap for two parallel p-orbitals given the atomic numbers and distance of separation

PARAMYZE Subroutine

"paramyze" prints the force field parameters used in the computation of each of the potential energy terms

PARTYZE Subroutine

"partyze" prints the energy component and number of interactions for each of the potential energy terms

PASSB Subroutine

PASSB2 Subroutine

PASSB3 Subroutine

PASSB4 Subroutine

PASSB5 Subroutine

PASSF Subroutine

PASSF2 Subroutine

PASSF3 Subroutine

PASSF4 Subroutine

PASSF5 Subroutine

PATH Program

"path" locates a series of structures equally spaced along a conformational pathway connecting the input reactant and product structures; a series of constrained optimizations orthogonal to the path is done via Lagrangian multipliers

PATH1 Function

PATHPNT Subroutine

"pathpnt" finds a structure on the synchronous transit path with the specified path value "tpath"

PATHSCAN Subroutine

"pathscan" makes a scan of a synchronous transit pathway by computing structures and energies for specific path values

PATHVAL Subroutine

"pathval" computes the synchronous transit path value for the specified structure

PAULING Subroutine

"pauling" uses a rigid body optimization to approximately pack multiple polypeptide chains

PAULING1 Function

"pauling1" is a service routine that computes the energy and gradient for optimally conditioned variable metric optimization of rigid bodies

PBDIRECTPOLFORCE Subroutine

PBEMPOLE Subroutine

"pbempole" calculates the permanent multipole PB energy, field, forces and torques

PBMUTUALPOLFORCE Subroutine

PDBATOM Subroutine

"pdbatom" adds an atom to the Protein Data Bank file

PDBXYZ Program

"pdbxyz" takes as input a Protein Data Bank file and then converts to and writes out a Cartesian coordinates file and, for biopolymers, a sequence file

PIALTER Subroutine

"pialter" modifies bond lengths and force constants according to the "planar" P-P-P bond order values; also alters 2-fold torsional parameters based on the "nonplanar" bond orders

PICALC Subroutine

"picalc" performs a modified Pariser-Parr-Pople molecular orbital calculation for each conjugated pisystem

PIMOVE Subroutine

"pimove" rotates the vector between atoms "list(1)" and "list(2)" so that atom 1 is at the origin and atom 2 along the x-axis; the atoms defining the respective planes are also moved and their bond lengths normalized

PIPLANE Subroutine

"piplane" selects the three atoms which specify the plane perpendicular to each p-orbital; the current version will fail in certain situations, including ketenes, allenes, and isolated or adjacent triple bonds

PISCF Subroutine

"piscf" performs an SCF molecular orbital calculation for a pisystem to determine bond orders used in parameter scaling

PITILT Subroutine

"pitilt" calculates for each pibond the ratio of the actual p-orbital overlap integral to the ideal overlap if the same orbitals were perfectly parallel

PLACE Subroutine

"place" finds the probe sites by putting the probe sphere tangent to each triple of neighboring atoms

PMONTE Subroutine

"pmonte" implements a Monte Carlo barostat via random trial changes in the periodic box volume and shape

POLARGRP Subroutine

"polargrp" generates members of the polarization group of each atom and separate lists of the 1-2, 1-3 and 1-4 group connectivities

POLARIZE Program

"polarize" computes the molecular polarizability by applying an external field along each axis followed by diagonalization of the resulting polarizability tensor

POLEDIT Program

"poledit" provides for the modification and manipulation of polarizable atomic multipole electrostatic models

POLESORT Subroutine

"polesort" sorts a set of atomic multipole parameters based on the atom types of centers involved

POLYMER Subroutine

"polymer" tests for the presence of an infinite polymer extending across periodic boundaries

POLYP Subroutine

"polyp" is a polynomial product routine that multiplies two algebraic forms

POTENTIAL Program

"potential" calculates the electrostatic potential for a molecule at a set of grid points; optionally compares to a target potential or optimizes electrostatic parameters

POTGRID Subroutine

"potgrid" generates electrostatic potential grid points in radially distributed shells based on the molecular surface

POTNRG Function

POTOFF Subroutine

"potoff" clears the forcefield definition by turning off the use of each of the potential energy functions

POTPOINT Subroutine

"potpoint" calculates the electrostatic potential at a grid point "i" as the total electrostatic interaction energy of the system with a positive charge located at the grid point

POTSTAT Subroutine

"potstat" computes and prints statistics for the electrostatic potential over a set of grid points

POTWRT Subroutine

PRECONBLK Subroutine

"preconblk" applies a preconditioner to an atom block section of the Hessian matrix

PRECOND Subroutine

"precond" solves a simplified version of the Newton equations Ms = r, and uses the result to precondition linear conjugate gradient iterations on the full Newton equations in "tnsolve"

PRESSURE Subroutine

"pressure" uses the internal virial to find the pressure in a periodic box and maintains a constant desired pressure via a barostat method

PRESSURE2 Subroutine

"pressure2" applies a box size and velocity correction at the half time step as needed for the Monte Carlo barostat

PRIORITY Function

"priority" decides which of a set of connected atoms should have highest priority in construction of a local coordinate frame and returns its atom number; if all atoms are of equal priority then zero is returned

PRMEDIT Program

"prmedit" reformats an existing parameter file, and revises type and class numbers based on the "atom" parameter ordering

PRMFORM Subroutine

"prmform" formats each individual parameter record to conform to a consistent text layout

PRMKEY Subroutine

"prmkey" parses a text string to extract keywords related to force field potential energy functional forms and constants

PRMORDER Subroutine

"prmorder" places a list of atom type or class numbers into canonical order for potential energy parameter definitions

PRMSORT Subroutine

"prmsort" places a list of atom type or class numbers into canonical order for potential energy parameter definitions

PRMVAR Subroutine

"prmvar" determines the optimization values from the corresponding electrostatic potential energy parameters

PRMVAR Subroutine

"prmvar" determines the optimization values from the corresponding valence potential energy parameters

PROCHAIN Subroutine

"prochain" builds up the internal coordinates for an amino acid sequence from the phi, psi, omega and chi values

PROJCT Subroutine

PROJECT Subroutine

"project" reads locked vectors from a binary file and projects them out of the components of the set of trial eigenvectors using the relation Y = X - U * U^T * X

PROJECTK Subroutine

"projectk" reads locked vectors from a binary file and projects them out of the components of the set of trial eigenvectors using the relation Y = X - U * U^T * X

PROMO Subroutine

"promo" writes a banner message containing information about the Tinker version, release date and copyright notice

PROPERTY Function

"property" takes two input snapshot frames and computes the value of the property for which the correlation function is being accumulated

PROSIDE Subroutine

"proside" builds the side chain for a single amino acid residue in terms of internal coordinates

PROTEIN Program

"protein" builds the internal and Cartesian coordinates of a polypeptide from amino acid sequence and torsional angle values for the peptide backbone and side chains

PRTARC Subroutine

"prtarc" writes out a set of Cartesian coordinates for all active atoms in the Tinker XYZ archive format

PRTDYN Subroutine

"prtdyn" writes out the information needed to restart a molecular dynamics trajectory to an external disk file

PRTERR Subroutine

"prterr" writes out a set of coordinates to a disk file prior to aborting on a serious error

PRTFIT Subroutine

"prtfit" makes a key file containing results from fitting a charge or multipole model to an electrostatic potential grid

PRTINT Subroutine

"prtint" writes out a set of Z-matrix internal coordinates to an external disk file

PRTMOD Subroutine

"prtmod" writes out a set of modified Cartesian coordinates with an optional atom number offset to an external disk file

PRTMOL2 Program

"prtmol2" writes out a set of coordinates in Tripos MOL2 format to an external disk file

PRTPDB Subroutine

"prtpdb" writes out a set of Protein Data Bank coordinates to an external disk file

PRTPOLE Subroutine

"prtpole" creates a coordinates file, and a key file with atomic multipoles corrected for intergroup polarization

PRTPRM Subroutine

"prtprm" writes out a formatted listing of the default set of potential energy parameters for a force field

PRTSEQ Subroutine

"prtseq" writes out a biopolymer sequence to an external disk file with 15 residues per line and distinct chains separated by blank lines

PRTVAL Subroutine

"prtval" writes the final valence parameter results to the standard output and appends the values to a key file

PRTVIB Subroutine

"prtvib" writes to an external disk file a series of coordinate sets representing motion along a vibrational normal mode

PRTXYZ Subroutine

"prtxyz" writes out a set of Cartesian coordinates to an external disk file

PSCALE Subroutine

"pscale" implements a Berendsen barostat by scaling the coordinates and box dimensions via coupling to an external constant pressure bath

PSS Program

"pss" implements the potential smoothing plus search method for global optimization in Cartesian coordinate space with local searches performed in Cartesian or torsional space

PSS1 Function

"pss1" is a service routine that computes the energy and gradient during PSS global optimization in Cartesian coordinate space

PSS2 Subroutine

"pss2" is a service routine that computes the sparse matrix Hessian elements during PSS global optimization in Cartesian coordinate space

PSSRGD1 Function

"pssrgd1" is a service routine that computes the energy and gradient during PSS global optimization over rigid bodies

PSSRIGID Program

"pssrigid" implements the potential smoothing plus search method for global optimization for a set of rigid bodies

PSSROT Program

"pssrot" implements the potential smoothing plus search method for global optimization in torsional space

PSSROT1 Function

"pssrot1" is a service routine that computes the energy and gradient during PSS global optimization in torsional space

PSSWRITE Subroutine

PTEST Subroutine

"ptest" determines the numerical virial tensor, and compares analytical to numerical values for dE/dV and isotropic pressure

PTINCY Function

PZEXTR Subroutine

"pzextr" is a polynomial extrapolation routine used during Bulirsch-Stoer integration of ordinary differential equations

QIROTMAT Subroutine

"qirotmat" finds a rotation matrix that describes the interatomic vector

QONVEC Subroutine

"qonvec" is a vector utility routine used during sliding block iterative matrix diagonalization

QRFACT Subroutine

"qrfact" computes the QR factorization of an m by n matrix a via Householder transformations with optional column pivoting; the routine determines an orthogonal matrix q, a permutation matrix p, and an upper trapezoidal matrix r with diagonal elements of nonincreasing magnitude, such that a*p = q*r; the Householder transformation for column k, k = 1,2,...,min(m,n), is of the form:

QRSOLVE Subroutine

"qrsolve" solves a*x = b and d*x = 0 in the least squares sense; used with routine "qrfact" to solve least squares problems

QUATFIT Subroutine

"quatfit" uses a quaternion-based method to achieve the best fit superposition of two sets of coordinates

RADIAL Program

"radial" finds the radial distribution function for a specified pair of atom types via analysis of a set of coordinate frames

RANDOM Function

"random" generates a random number on [0,1] via a long period generator due to L'Ecuyer with Bays-Durham shuffle

RANVEC Subroutine

"ranvec" generates a unit vector in 3-dimensional space with uniformly distributed random orientation

RATTLE Subroutine

"rattle" implements the first portion of the RATTLE algorithm by correcting atomic positions and half-step velocities to maintain interatomic distance and absolute spatial constraints

RATTLE2 Subroutine

"rattle2" implements the second portion of the RATTLE algorithm by correcting the full-step velocities in order to maintain interatomic distance constraints

READBLK Subroutine

"readblk" reads in a set of snapshot frames and transfers the values to internal arrays for use in the computation of time correlation functions

READDYN Subroutine

"readdyn" get the positions, velocities and accelerations for a molecular dynamics restart from an external disk file

READGARC Subroutine

"readgarc" reads data from Gaussian archive section; each entry is terminated with a backslash symbol

READGAU Subroutine

"readgau" reads an ab initio optimized structure, forces, Hessian and frequencies from a Gaussian 09 output file

READGDMA Subroutine

"readgdma" takes the DMA output in spherical harmonics from the GDMA program and converts to Cartesian multipoles in the global coordinate frame

READINT Subroutine

"readint" gets a set of Z-matrix internal coordinates from an external file

READMOL Subroutine

"readmol" gets a set of MDL MOL coordinates from an external disk file

READMOL2 Subroutine

"readmol2" gets a set of Tripos MOL2 coordinates from an external disk file

READPDB Subroutine

"readpdb" gets a set of Protein Data Bank coordinates from an external disk file

READPOT Subroutine

"readpot" gets a set of grid points and target electrostatic potential values from an external disk file

READPRM Subroutine

"readprm" processes the potential energy parameter file in order to define the default force field parameters

READSEQ Subroutine

"readseq" gets a biopolymer sequence containing one or more separate chains from an external file; all lines containing sequence must begin with the starting sequence number, the actual sequence is read from subsequent nonblank characters

READXYZ Subroutine

"readxyz" gets a set of Cartesian coordinates from an external disk file

REFINE Subroutine

"refine" performs minimization of the atomic coordinates of an initial crude embedded distance geometry structure versus the bound, chirality, planarity and torsional error functions

RELEASEMONITOR Subroutine

REPLICA Subroutine

"replica" decides between images and replicates for generation of periodic boundary conditions, and sets the cell replicate list if the replicates method is to be used

RESPA Subroutine

"respa" performs a single multiple time step molecular dynamics step using the reversible reference system propagation algorithm (r-RESPA) via a Verlet core with the potential split into fast- and slow-evolving portions

RFINDEX Subroutine

"rfindex" finds indices for each multipole site for use in computing reaction field energetics

RGDSTEP Subroutine

"rgdstep" performs a single molecular dynamics time step via a rigid body integration algorithm

RIBOSOME Subroutine

"ribosome" translates a polypeptide structure in Protein Data Bank format to a Cartesian coordinate file and sequence file

RIGIDXYZ Subroutine

"rigidxyz" computes Cartesian coordinates for a rigid body group via rotation and translation of reference coordinates

RINGS Subroutine

"rings" searches the structure for small rings and stores their constituent atoms, and optionally reduces large rings into their component smaller rings

RMSERROR Subroutine

"rmserror" computes the maximum absolute deviation and the rms deviation from the distance bounds, and the number and rms value of the distance restraint violations

RMSFIT Function

"rmsfit" computes the rms fit of two coordinate sets

ROTANG Function

ROTCHECK Function

"rotcheck" tests a specified candidate rotatable bond for the disallowed case where inactive atoms are found on both sides of the candidate bond

ROTEULER Subroutine

"roteuler" computes a set of Euler angle values consistent with an input rotation matrix

ROTFRAME Subroutine

"rotframe" takes the global multipole moments and rotates them into the local coordinate frame defined at each atomic site

ROTLIST Subroutine

"rotlist" generates the minimum list of all the atoms lying to one side of a pair of directly bonded atoms; optionally finds the minimal list by choosing the side with fewer atoms

ROTMAT Subroutine

"rotmat" finds the rotation matrix that rotates the local coordinate system into the global frame at a multipole site

ROTPOLE Subroutine

"rotpole" constructs the set of atomic multipoles in the global frame by applying the correct rotation matrix for each site

ROTRGD Subroutine

"rotrgd" finds the rotation matrix for a rigid body due to a single step of dynamics

ROTSITE Subroutine

"rotsite" rotates the local frame atomic multipoles at a specified site into the global coordinate frame by applying a rotation matrix

SADDLE Program

"saddle" finds a transition state between two conformational minima using a combination of ideas from the synchronous transit (Halgren-Lipscomb) and quadratic path (Bell-Crighton) methods

SADDLE1 Function

"saddle1" is a service routine that computes the energy and gradient for transition state optimization

SADDLES Subroutine

"saddles" constructs circles, convex edges and saddle faces

SAVEYZE Subroutine

"saveyze" prints the atomic forces and/or the induced dipoles to separate external disk files

SBGUESS Subroutine

"sbguess" sets approximate stretch-bend force constants based on atom type and connected atoms

SCAN Program

"scan" attempts to find all the local minima on a potential energy surface via an iterative series of local searches along normal mode directions

SCAN1 Function

"scan1" is a service routine that computes the energy and gradient during exploration of a potential energy surface via iterative local search

SCAN2 Subroutine

"scan2" is a service routine that computes the sparse matrix Hessian elements during exploration of a potential energy surface via iterative local search

SCANPDB Subroutine

"scanpdb" reads the first model in a Protein Data Bank file and sets chains, alternate sites and insertion records to be used

SDAREA Subroutine

"sdarea" optionally scales the atomic friction coefficient of each atom based on its accessible surface area

SDSTEP Subroutine

"sdstep" performs a single stochastic dynamics time step via the velocity Verlet integration algorithm

SDTERM Subroutine

"sdterm" finds the frictional and random terms needed to update positions and velocities during stochastic dynamics

SEARCH Subroutine

"search" is a unidimensional line search based upon parabolic extrapolation and cubic interpolation using both function and gradient values

SETACCELERATION Subroutine

SETATOMIC Subroutine

SETATOMTYPES Subroutine

SETCHARGE Subroutine

SETCHUNK Subroutine

"setchunk" marks a chunk in the PME spatial table which is overlapped by the B-splines for a site

SETCONNECTIVITY Subroutine

SETCOORDINATES Subroutine

SETELECT Subroutine

"setelect" assigns partial charge, bond dipole and atomic multipole parameters for the current structure, as needed for computation of the electrostatic potential

SETENERGY Subroutine

SETFILE Subroutine

SETFORCEFIELD Subroutine

SETFRAME Subroutine

"setframe" assigns a local coordinate frame at each atomic multipole site using high priority connected atoms along axes

SETGRADIENTS Subroutine

SETINDUCED Subroutine

SETKEYWORD Subroutine

SETMASS Subroutine

SETMDTIME Subroutine

SETMOL2 Program

"setmol2" assigns MOL2 atom names/types/charges and bond types based upon atomic numbers and connectivity

SETNAME Subroutine

SETPAIR Program

"setpair" is a service routine that assigns flags, sets cutoffs and allocates arrays used by different pairwise neighbor methods

SETPOLAR Subroutine

"setpolar" assigns atomic polarizabilities, Thole damping or charge penetration parameters, and polarization groups with user modification of these values

SETSTEP Subroutine

SETSTORY Subroutine

SETTIME Subroutine

"settime" initializes the wall clock and elapsed CPU times

SETUPDATED Subroutine

SETVELOCITY Subroutine

SHAKE Subroutine

"shake" implements the SHAKE algorithm by correcting atomic positions to maintain interatomic distance and absolute spatial constraints

SHAKEF Subroutine

"shakef" modifies the gradient to remove components along any holonomic distance contraints using a variant of SHAKE

SHAKEUP Subroutine

"shakeup" initializes any holonomic constraints for use with the SHAKE and RATTLE algorithms

SHROTMAT Subroutine

"shrotmat" finds the rotation matrix that converts spherical harmonic quadrupoles from the local to the global frame given the required dipole rotation matrix

SHROTSITE Subroutine

"shrotsite" converts spherical harmonic multipoles from the local to the global frame given required rotation matrices

SIGMOID Function

"sigmoid" implements a normalized sigmoidal function on the interval [0,1]; the curves connect (0,0) to (1,1) and have a cooperativity controlled by beta, they approach a straight line as beta -> 0 and get more nonlinear as beta increases

SIMPLEX Subroutine

"simplex" is a general multidimensional Nelder-Mead simplex optimization routine requiring only repeated evaluations of the objective function

SIMPLEX1 Function

"simplex1" is a service routine used only by the Nelder-Mead simplex optimization method

SKTDYN Subroutine

"sktdyn" sends the current dynamics info via a socket

SKTINIT Subroutine

"sktinit" sets up socket communication with the graphical user interface by starting a Java virtual machine, initiating a server, and loading an object with system information

SKTKILL Subroutine

"sktkill" closes the server and Java virtual machine

SKTOPT Subroutine

"sktopt" sends the current optimization info via a socket

SLATER Subroutine

"slater" is a general routine for computing the overlap integrals between two Slater-type orbitals

SNIFFER Program

"sniffer" performs a global energy minimization using a discrete version of Griewank's global search trajectory

SNIFFER1 Function

"sniffer1" is a service routine that computes the energy and gradient for the Sniffer global optimization method

SOAK Subroutine

"soak" takes a currently defined solute system and places it into a solvent box, with removal of any solvent molecules that overlap the solute

SORT Subroutine

"sort" takes an input list of integers and sorts it into ascending order using the Heapsort algorithm

SORT10 Subroutine

"sort10" takes an input list of character strings and sorts it into alphabetical order using the Heapsort algorithm, duplicate values are removed from the final sorted list

SORT2 Subroutine

"sort2" takes an input list of reals and sorts it into ascending order using the Heapsort algorithm; it also returns a key into the original ordering

SORT3 Subroutine

"sort3" takes an input list of integers and sorts it into ascending order using the Heapsort algorithm; it also returns a key into the original ordering

SORT4 Subroutine

"sort4" takes an input list of integers and sorts it into ascending absolute value using the Heapsort algorithm

SORT5 Subroutine

"sort5" takes an input list of integers and sorts it into ascending order based on each value modulo "m"

SORT6 Subroutine

"sort6" takes an input list of character strings and sorts it into alphabetical order using the Heapsort algorithm

SORT7 Subroutine

"sort7" takes an input list of character strings and sorts it into alphabetical order using the Heapsort algorithm; it also returns a key into the original ordering

SORT8 Subroutine

"sort8" takes an input list of integers and sorts it into ascending order using the Heapsort algorithm, duplicate values are removed from the final sorted list

SORT9 Subroutine

"sort9" takes an input list of reals and sorts it into ascending order using the Heapsort algorithm, duplicate values are removed from the final sorted list

SPACEFILL Program

"spacefill" computes the surface area and volume of a structure; the van der Waals, accessible-excluded, and contact-reentrant definitions are available

SPECTRUM Program

"spectrum" computes a power spectrum over a wavelength range from the velocity autocorrelation as a function of time

SPHERE Subroutine

"sphere" finds a specified number of uniformly distributed points on a sphere of unit radius centered at the origin

SQUARE Subroutine

"square" is a nonlinear least squares routine derived from the IMSL BCLSF routine and the MINPACK LMDER routine; the Jacobian is estimated by finite differences and bounds can be specified for the variables to be refined

SUFFIX Subroutine

"suffix" checks a filename for the presence of an extension, and appends an extension and version if none is found

SUPERPOSE Program

"superpose" takes pairs of structures and superimposes them in the optimal least squares sense; it will attempt to match all atom pairs or only those specified by the user

SURFACE Subroutine

"surface" performs an analytical computation of the weighted solvent accessible surface area of each atom and the first derivatives of the area with respect to Cartesian coordinates

SURFACE1 Subroutine

"surface1" performs an analytical computation of the weighted solvent accessible surface area of each atom and the first derivatives of the area with respect to Cartesian coordinates

SURFATOM Subroutine

"surfatom" performs an analytical computation of the surface area of a specified atom; a simplified version of "surface"

SURFATOM1 Subroutine

"surfatom1" performs an analytical computation of the surface area and first derivatives with respect to Cartesian coordinates of a specified atom

SWITCH Subroutine

"switch" sets the coeffcients used by the fifth and seventh order polynomial switching functions for spherical cutoffs

SYMMETRY Subroutine

"symmetry" applies symmetry operators to the fractional coordinates of the asymmetric unit in order to generate the symmetry related atoms of the full unit cell

SYSTYZE Subroutine

"systyze" is an auxiliary routine for the analyze program that prints general information about the molecular system and the force field model

TABLE_FILL Subroutine

"table_fill" constructs an array which stores the spatial regions of the particle mesh Ewald grid with contributions from each site

TANGENT Subroutine

"tangent" finds the projected gradient on the synchronous transit path for a point along the transit pathway

TCGSWAP Subroutine

"tcgswap" switches two sets of induced dipole quantities for use with the TCG induced dipole solver

TCG_ALPHA12 Subroutine

"tcg_alpha12" computes source1 = alpha*source1 and source2 = alpha*source2

TCG_ALPHA22 Subroutine

"tcg_alpha22" computes result1 = alpha*source1 and result2 = alpha*source2

TCG_ALPHAQUAD Subroutine

"tcg_alphaquad" computes the quadratic form, <a*alpha*b>, where alpha is the diagonal atomic polarizability matrix

TCG_DOTPROD Subroutine

"tcg_dotprod" computes the dot product of two vectors of length n elements

TCG_RESOURCE Subroutine

"tcg_resource" sets the number of mutual induced dipole pairs based on the passed argument

TCG_T0 Subroutine

"tcg_t0" applies T matrix to ind/p, and returns v3d/p T = 1/alpha + Tu

TCG_UFIELD Subroutine

"tcg_ufield" applies -Tu to ind/p and returns v3d/p

TCG_UPDATE Subroutine

"tcg_update" computes pvec = alpha*rvec + beta*pvec; if the preconditioner is not used, then alpha = identity

TEMPER Subroutine

"temper" computes the instantaneous temperature and applies a thermostat via Berendsen or Bussi-Parrinello velocity scaling, Andersen stochastic collisions or Nose-Hoover chains; also uses Berendsen scaling for any iEL induced dipole variables

TEMPER2 Subroutine

"temper2" applies a velocity correction at the half time step as needed for the Nose-Hoover thermostat

TESTGRAD Program

"testgrad" computes and compares the analytical and numerical gradient vectors of the potential energy function with respect to Cartesian coordinates

TESTHESS Program

"testhess" computes and compares the analytical and numerical Hessian matrices of the potential energy function with respect to Cartesian coordinates

TESTPAIR Program

"testpair" performs a set of timing tests to compare the evaluation of potential energy and energy/gradient using different methods for finding pairwise neighbors

TESTPOL Program

"testpol" compares the induced dipoles from direct polarization, mutual SCF iterations, perturbation theory extrapolation (OPT), and truncated conjugate gradient (TCG) solvers

TESTROT Program

"testrot" computes and compares the analytical and numerical gradient vectors of the potential energy function with respect to rotatable torsional angles

TESTVIR Program

"testvir" computes the analytical internal virial and compares it to a numerical virial derived from the finite difference derivative of the energy with respect to lattice vectors

TIMER Program

"timer" measures the CPU time required for file reading and parameter assignment, potential energy computation, energy and gradient computation, and Hessian matrix evaluation

TIMEROT Program

"timerot" measures the CPU time required for file reading and parameter assignment, potential energy computation, energy and gradient over torsions, and torsional angle Hessian matrix evaluation

TNCG Subroutine

"tncg" implements a truncated Newton optimization algorithm in which a preconditioned linear conjugate gradient method is used to approximately solve Newton's equations; special features include use of an explicit sparse Hessian or finite-difference gradient-Hessian products within the PCG iteration; the exact Newton search directions can be used optionally; by default the algorithm checks for negative curvature to prevent convergence to a stationary point having negative eigenvalues; if a saddle point is desired this test can be removed by disabling "negtest"

TNSOLVE Subroutine

"tnsolve" uses a linear conjugate gradient method to find an approximate solution to the set of linear equations represented in matrix form by Hp = -g (Newton's equations)

TORFIT1 Function

"torfit1" is a service routine that computes the energy and gradient for a low storage BFGS optimization in Cartesian coordinate space

TORGUESS Subroutine

"torguess" set approximate torsion amplitude parameters based on atom type and connected atoms

TORPHASE Subroutine

"torphase" sets the n-fold amplitude and phase values for each torsion via sorting of the input parameters

TORQUE Subroutine

"torque" takes the torque values on a single site defined by a local coordinate frame and converts to Cartesian forces on the original site and sites specifying the local frame, also gives the x,y,z-force components needed for virial computation

TORSER Function

"torser" computes the torsional error function and its first derivatives with respect to the atomic Cartesian coordinates based on the deviation of specified torsional angles from desired values, the contained bond angles are also restrained to avoid a numerical instability

TORSFIT Program

"torsfit" refines torsional force field parameters based on a quantum mechanical potential surface and analytical gradient

TORSIONS Subroutine

"torsions" finds the total number of torsional angles and the numbers of the four atoms defining each torsional angle

TORUS Subroutine

"torus" sets a list of all of the temporary torus positions by testing for a torus between each atom and its neighbors

TOTERR Function

"toterr" is the error function and derivatives for a distance geometry embedding; it includes components from the distance bounds, hard sphere contacts, local geometry, chirality and torsional restraint errors

TRANSFORM Subroutine

"transform" diagonalizes the current basis vectors to produce trial roots for sliding block iterative matrix diagonalization

TRANSIT Function

"transit" evaluates the synchronous transit function and gradient; linear and quadratic transit paths are available

TRBASIS Subroutine

"trbasis" forms translation and rotation basis vectors used during vibrational analysis via block iterative diagonalization

TRIANGLE Subroutine

"triangle" smooths the upper and lower distance bounds via the triangle inequality using a full-matrix variant of the Floyd-Warshall shortest path algorithm; this routine is usually much slower than the sparse matrix shortest path methods in "geodesic" and "trifix", and should be used only for comparison with answers generated by those routines

TRIFIX Subroutine

"trifix" rebuilds both the upper and lower distance bound matrices following tightening of one or both of the bounds between a specified pair of atoms, "p" and "q", using a modification of Murchland's shortest path update algorithm

TRIGGER Subroutine

"trigger" constructs a set of initial trial vectors for use during sliding block iterative matrix diagonalization

TRIMHEAD Subroutine

"trimhead" removes blank spaces before the first non-blank character in a text string by shifting the string to the left

TRIMTEXT Function

"trimtext" finds and returns the location of the last non-blank character before the first null character in an input text string; the function returns zero if no such character is found

TRIPLE Function

"triple" finds the triple product of three vectors; used as a service routine by the Connolly surface area and volume computation

TRUST Subroutine

"trust" updates the model trust region for a nonlinear least squares calculation based on ideas found in NL2SOL and Dennis and Schnabel's book

UBUILD Subroutine

"ubuild" performs a complete rebuild of the polarization preconditioner neighbor list for all sites

UDIRECT1 Subroutine

"udirect1" computes the reciprocal space contribution of the permanent atomic multipole moments to the field

UDIRECT2A Subroutine

"udirect2a" computes the real space contribution of the permanent atomic multipole moments to the field via a double loop

UDIRECT2B Subroutine

"udirect2b" computes the real space contribution of the permanent atomic multipole moments to the field via a neighbor list

UFIELD0A Subroutine

"ufield0a" computes the mutual electrostatic field due to induced dipole moments via a double loop

UFIELD0B Subroutine

"ufield0b" computes the mutual electrostatic field due to induced dipole moments via a pair list

UFIELD0C Subroutine

"ufield0c" computes the mutual electrostatic field due to induced dipole moments via Ewald summation

UFIELD0D Subroutine

"ufield0d" computes the mutual electrostatic field due to induced dipole moments for use with with generalized Kirkwood implicit solvation

UFIELD0E Subroutine

"ufield0e" computes the mutual electrostatic field due to induced dipole moments via a Poisson-Boltzmann solver

UFIELDI Subroutine

"ufieldi" computes the electrostatic field due to intergroup induced dipole moments

ULIGHT Subroutine

"ulight" performs a complete rebuild of the polarization preconditioner pair neighbor list for all sites using the method of lights

ULIST Subroutine

"ulist" performs an update or a complete rebuild of the neighbor lists for the polarization preconditioner

ULSPRED Subroutine

"ulspred" uses standard extrapolation or a least squares fit to set coefficients of an induced dipole predictor polynomial

UMUTUAL1 Subroutine

"umutual1" computes the reciprocal space contribution of the induced atomic dipole moments to the field

UMUTUAL2A Subroutine

"umutual2a" computes the real space contribution of the induced atomic dipole moments to the field via a double loop

UMUTUAL2B Subroutine

"umutual2b" computes the real space contribution of the induced atomic dipole moments to the field via a neighbor list

UNITCELL Subroutine

"unitcell" gets the periodic boundary box size and related values from an external keyword file

UPCASE Subroutine

"upcase" converts a text string to all upper case letters

URYGUESS Function

"uryguess" sets approximate Urey-Bradley force constants based on atom type and connected atoms

USCALE0A Subroutine

"uscale0a" builds and applies a preconditioner for the conjugate gradient induced dipole solver using a double loop

USCALE0B Subroutine

"uscale0b" builds and applies a preconditioner for the conjugate gradient induced dipole solver using a neighbor pair list

VALENCE Program

"valence" refines force field parameters for valence terms based on a quantum mechanical optimized structure and frequencies

VALENCE1 Function

"valence1" is a service routine that computes the energy and gradient for a structure during valence parameter fitting

VALFIT1 Function

"valfit1" is a service routine that computes the RMS error and gradient for valence parameters fit to QM results

VALGUESS Subroutine

"valguess" sets approximate valence parameter values based on quantum mechanical structure and frequency data

VALRMS Function

"valrms" evaluates a valence parameter goodness-of-fit error function based on comparison of forces, frequencies, bond lengths and angles to QM results

VAM Subroutine

"vam" takes the analytical molecular surface defined as a collection of spherical and toroidal polygons and uses it to compute the volume and surface area

VARPRM Subroutine

"varprm" copies the current optimization values into the corresponding electrostatic potential energy parameters

VARPRM Subroutine

"varprm" copies the current optimization values into the corresponding valence potential energy parameters

VBUILD Subroutine

"vbuild" performs a complete rebuild of the van der Waals pair neighbor list for all sites

VCROSS Subroutine

"vcross" finds the cross product of two vectors

VDWERR Function

"vdwerr" is the hard sphere van der Waals bound error function and derivatives that penalizes close nonbonded contacts, pairwise neighbors are generated via the method of lights

VDWGUESS Subroutine

"vdwguess" sets initial VDW parameters based on atom type and connected atoms

VECANG Function

"vecang" finds the angle between two vectors handed with respect to a coordinate axis; returns an angle in the range [0,2*pi]

VERLET Subroutine

"verlet" performs a single molecular dynamics time step via the velocity Verlet multistep recursion formula

VERSION Subroutine

"version" checks the name of a file about to be opened; if if "old" status is passed, the name of the highest current version is returned; if "new" status is passed the filename of the next available unused version is generated

VIBBIG Program

"vibbig" performs large-scale vibrational mode analysis using only vector storage and gradient evaluations; preconditioning is via an approximate inverse from a block diagonal Hessian, and a sliding block method is used to converge any number of eigenvectors starting from either lowest or highest frequency

VIBRATE Program

"vibrate" performs a vibrational normal mode analysis; the Hessian matrix of second derivatives is determined and then diagonalized both directly and after mass weighting; output consists of the eigenvalues of the force constant matrix as well as the vibrational frequencies and displacements

VIBROT Program

"vibrot" computes the eigenvalues and eigenvectors of the torsional Hessian matrix

VIRIYZE Subroutine

"propyze" finds and prints the internal virial, the dE/dV value and an estimate of the pressure

VLIGHT Subroutine

"vlight" performs a complete rebuild of the van der Waals pair neighbor list for all sites using the method of lights

VLIST Subroutine

"vlist" performs an update or a complete rebuild of the nonbonded neighbor lists for vdw sites

VNORM Subroutine

"vnorm" normalizes a vector to unit length; used as a service routine by the Connolly surface area and volume computation

VOLUME Subroutine

"volume" calculates the excluded volume via the Connolly analytical volume and surface area algorithm

VOLUME1 Subroutine

"volume1" calculates first derivatives of the total excluded volume with respect to the Cartesian coordinates of each atom

VOLUME2 Subroutine

"volume2" calculates second derivatives of the total excluded volume with respect to the Cartesian coordinates of the atoms

WATSON Subroutine

"watson" uses a rigid body optimization to approximately align the paired strands of a nucleic acid double helix

WATSON1 Function

"watson1" is a service routine that computes the energy and gradient for optimally conditioned variable metric optimization of rigid bodies

WIGGLE Subroutine

"wiggle" applies a random perturbation to the atomic coordinates to avoid numerical instabilities for various linear, planar and symmetric structures

XTALERR Subroutine

"xtalerr" computes an error function value derived from lattice energies, dimer intermolecular energies and the gradient with respect to structural parameters

XTALFIT Program

"xtalfit" determines optimized van der Waals and electrostatic parameters by fitting to crystal structures, lattice energies, and dimer structures and interaction energies

XTALMIN Program

"xtalmin" performs a full crystal energy minimization by optimizing over fractional atomic coordinates and the six lattice lengths and angles

XTALMIN1 Function

"xtalmin1" is a service routine that computes the energy and gradient with respect to fractional coordinates and lattice dimensions for a crystal energy minimization

XTALMOVE Subroutine

"xtalmove" converts fractional to Cartesian coordinates for rigid molecules during optimization of force field parameters

XTALPRM Subroutine

"xtalprm" stores or retrieves a molecular structure; used to make a previously stored structure the active structure, or to store a structure for later use

XTALWRT Subroutine

"xtalwrt" prints intermediate results during fitting of force field parameters to structures and energies

XYZATM Subroutine

"xyzatm" computes the Cartesian coordinates of a single atom from its defining internal coordinate values

XYZEDIT Program

"xyzedit" provides for modification and manipulation of the contents of Cartesian coordinates files

XYZINT Program

"xyzint" takes as input a Cartesian coordinates file, then converts to and writes out an internal coordinates file

XYZMOL2 Program

"xyzmol2" takes as input a Cartesian coordinates file, converts to and then writes out a Tripos MOL2 file

XYZPDB Program

"xyzpdb" takes as input a Cartesian coordinates file, then converts to and writes out a Protein Data Bank file

XYZRIGID Subroutine

"xyzrigid" computes the center of mass and Euler angle rigid body coordinates for each atom group in the system

ZATOM Subroutine

"zatom" adds an atom to the end of the current Z-matrix and then increments the atom counter; atom type, defining atoms and internal coordinates are passed as arguments

ZHELP Subroutine

"zhelp" prints the general information and instructions for the Z-matrix editing program

ZVALUE Subroutine

"zvalue" gets user supplied values for selected coordinates as needed by the internal coordinate editing program

CHAPTER

TEN

MODULES & GLOBAL VARIABLES

The Fortran modules found in the Tinker package are listed below along with a brief description of the variables associated with each module. Each individual module contains a set of globally allocated variables available to any program unit upon inclusion of that module. A source listing containing each of the Tinker functions and subroutines and its included modules can be produced by running the "listing.make" script found in the distribution.

ACTION Module **total number of each energy term type**

```
neb             number of bond stretch energy terms computed
nea             number of angle bend energy terms computed
neba            number of stretch-bend energy terms computed
neub            number of Urey-Bradley energy terms computed
neaa            number of angle-angle energy terms computed
neopb           number of out-of-plane bend energy terms computed
neopd           number of out-of-plane distance energy terms computed
neid            number of improper dihedral energy terms computed
neit            number of improper torsion energy terms computed
net             number of torsional energy terms computed
nept            number of pi-system torsion energy terms computed
nebt            number of stretch-torsion energy terms computed
neat            number of angle-torsion energy terms computed
nett            number of torsion-torsion energy terms computed
nev             number of van der Waals energy terms computed
ner             number of Pauli repulsion energy terms computed
nedsp           number of dispersion energy terms computed
nec             number of charge-charge energy terms computed
necd            number of charge-dipole energy terms computed
ned             number of dipole-dipole energy terms computed
nem             number of multipole energy terms computed
nep             number of polarization energy terms computed
nect            number of charge transfer energy terms computed
new             number of Ewald summation energy terms computed
nerxf           number of reaction field energy terms computed
nes             number of solvation energy terms computed
nelf            number of metal ligand field energy terms computed
neg             number of geometric restraint energy terms computed
nex             number of extra energy terms computed
```

ALIGN Module **information for structure superposition**

nfit	number of atoms to use in superimposing two structures
ifit	atom numbers of pairs of atoms to be superimposed
wfit	weights assigned to atom pairs during superposition

ANALYZ Module **energy components partitioned to atoms**

aesum	total potential energy partitioned over atoms
aeb	bond stretch energy partitioned over atoms
aea	angle bend energy partitioned over atoms
aeba	stretch-bend energy partitioned over atoms
aeub	Urey-Bradley energy partitioned over atoms
aeaa	angle-angle energy partitioned over atoms
aeopb	out-of-plane bend energy partitioned over atoms
aeopd	out-of-plane distance energy partitioned over atoms
aeid	improper dihedral energy partitioned over atoms
aeit	improper torsion energy partitioned over atoms
aet	torsional energy partitioned over atoms
aept	pi-system torsion energy partitioned over atoms
aebt	stretch-torsion energy partitioned over atoms
aeat	angle-torsion energy partitioned over atoms
aett	torsion-torsion energy partitioned over atoms
aev	van der Waals energy partitioned over atoms
aer	Pauli repulsion energy partitioned over atoms
aedsp	damped dispersion energy partitioned over atoms
aec	charge-charge energy partitioned over atoms
aecd	charge-dipole energy partitioned over atoms
aed	dipole-dipole energy partitioned over atoms
aem	multipole energy partitioned over atoms
aep	polarization energy partitioned over atoms
aect	charge transfer energy partitioned over atoms
aerxf	reaction field energy partitioned over atoms
aes	solvation energy partitioned over atoms
aelf	metal ligand field energy partitioned over atoms
aeg	geometric restraint energy partitioned over atoms
aex	extra energy term partitioned over atoms

ANGANG Module **angle-angles in current structure**

nangang	total number of angle-angle interactions
iaa	angle numbers used in each angle-angle term
kaa	force constant for angle-angle cross terms

ANGBND Module **bond angle bends in current structure**

nangle	total number of angle bends in the system
iang	numbers of the atoms in each angle bend
ak	harmonic angle force constant (kcal/mole/rad**2)
anat	ideal bond angle or phase shift angle (degrees)
afld	periodicity for Fourier angle bending term

ANGPOT Module **angle bend functional form details**

```
angunit         convert angle bending energy to kcal/mole
stbnunit        convert stretch-bend energy to kcal/mole
aaunit          convert angle-angle energy to kcal/mole
opbunit         convert out-of-plane bend energy to kcal/mole
opdunit         convert out-of-plane distance energy to kcal/mole
cang            cubic coefficient in angle bending potential
qang            quartic coefficient in angle bending potential
pang            quintic coefficient in angle bending potential
sang            sextic coefficient in angle bending potential
copb            cubic coefficient in out-of-plane bend potential
qopb            quartic coefficient in out-of-plane bend potential
popb            quintic coefficient in out-of-plane bend potential
sopb            sextic coefficient in out-of-plane bend potential
copd            cubic coefficient in out-of-plane distance potential
qopd            quartic coefficient in out-of-plane distance potential
popd            quintic coefficient in out-of-plane distance potential
sopd            sextic coefficient in out-of-plane distance potential
opbtyp          type of out-of-plane bend potential energy function
angtyp          type of angle bending function for each bond angle
```

ANGTOR Module angle-torsions in current structure

```
nangtor         total number of angle-torsion interactions
iat             torsion and angle numbers used in angle-torsion
kant            1-, 2- and 3-fold angle-torsion force constants
```

ARGUE Module command line arguments at run time

```
maxarg          maximum number of command line arguments
narg            number of command line arguments to the program
listarg         flag to mark available command line arguments
arg             strings containing the command line arguments
```

ASCII Module selected ASCII character code values

```
null            decimal value of ASCII code for null (0)
tab             decimal value of ASCII code for tab (9)
linefeed        decimal value of ASCII code for linefeed (10)
formfeed        decimal value of ASCII code for formfeed (12)
carriage        decimal value of ASCII code for carriage return (13)
escape          decimal value of ASCII code for escape (27)
space           decimal value of ASCII code for blank space (32)
exclamation     decimal value of ASCII code for exclamation (33)
quote           decimal value of ASCII code for double quote (34)
pound           decimal value of ASCII code for pound sign (35)
dollar          decimal value of ASCII code for dollar sign (36)
percent         decimal value of ASCII code for percent sign (37)
ampersand       decimal value of ASCII code for ampersand (38)
apostrophe      decimal value of ASCII code for single quote (39)
asterisk        decimal value of ASCII code for asterisk (42)
plus            decimal value of ASCII code for plus sign (43)
comma           decimal value of ASCII code for comma (44)
```

(continues on next page)

(continued from previous page)

```
minus           decimal value of ASCII code for minus sign (45)
period          decimal value of ASCII code for period (46)
frontslash      decimal value of ASCII codd for frontslash (47)
colon           decimal value of ASCII code for colon (58)
semicolon       decimal value of ASCII code for semicolon (59)
equal           decimal value of ASCII code for equal sign (61)
question        decimal value of ASCII code for question mark (63)
atsign          decimal value of ASCII code for at sign (64)
backslash       decimal value of ASCII code for backslash (92)
caret           decimal value of ASCII code for caret (94)
underbar        decimal value of ASCII code for underbar (95)
vertical        decimal value of ASCII code for vertical bar (124)
tilde           decimal value of ASCII code for tilde (126)
```

ATMLST Module local geometry indices for each atom

```
bndlist         list of the bond numbers involving each atom
anglist         list of the angle numbers centered on each atom
```

ATOMID Module atomic properties for current atoms

```
tag             integer atom labels from input coordinates file
class           atom class number for each atom in the system
atomic          atomic number for each atom in the system
valence         valence number for each atom in the system
mass            atomic weight for each atom in the system
name            atom name for each atom in the system
story           descriptive type for each atom in system
```

ATOMS Module number, position and type of atoms

```
n               total number of atoms in the current system
type            atom type number for each atom in the system
x               current x-coordinate for each atom in the system
y               current y-coordinate for each atom in the system
z               current z-coordinate for each atom in the system
```

BATH Module thermostat and barostat control values

```
maxnose         maximum length of Nose-Hoover thermostat chain
voltrial        mean number of steps between Monte Carlo moves
kelvin          target value for the system temperature (K)
atmsph          target value for the system pressure (atm)
tautemp         time constant for Berendsen thermostat (psec)
taupres         time constant for Berendsen barostat (psec)
compress        isothermal compressibility of medium (atm-1)
collide         collision frequency for Andersen thermostat
eta             velocity value for Bussi-Parrinello barostat
volmove         maximum volume move for Monte Carlo barostat (Ang**3)
vbar            velocity of log volume for Nose-Hoover barostat
qbar            mass of the volume for Nose-Hoover barostat
```

(continues on next page)

(continued from previous page)

```
gbar            force for the volume for Nose-Hoover barostat
vnh             velocity of each chained Nose-Hoover thermostat
qnh             mass for each chained Nose-Hoover thermostat
gnh             force for each chained Nose-Hoover thermostat
isothermal      logical flag governing use of temperature control
isobaric        logical flag governing use of pressure control
anisotrop       logical flag governing use of anisotropic pressure
thermostat      choice of temperature control method to be used
barostat        choice of pressure control method to be used
volscale        choice of scaling method for Monte Carlo barostat
```

BITOR Module bitorsions in the current structure

```
nbitor          total number of bitorsions in the system
ibitor          numbers of the atoms in each bitorsion
```

BNDPOT Module bond stretch functional form details

```
cbnd            cubic coefficient in bond stretch potential
qbnd            quartic coefficient in bond stretch potential
bndunit         convert bond stretch energy to kcal/mole
bndtyp          type of bond stretch potential energy function
```

BNDSTR Module bond stretches in the current structure

```
nbond           total number of bond stretches in the system
ibnd            numbers of the atoms in each bond stretch
bk              bond stretch force constants (kcal/mole/Ang**2)
bl              ideal bond length values in Angstroms
```

BOUND Module periodic boundary condition controls

```
polycut         cutoff distance for infinite polymer nonbonds
polycut2        square of infinite polymer nonbond cutoff
use_bounds      flag to use periodic boundary conditions
use_replica     flag to use replicates for periodic system
use_polymer     flag to mark presence of infinite polymer
```

BOXES Module periodic boundary condition parameters

```
xbox            length of a-axis of periodic box in Angstroms
ybox            length of b-axis of periodic box in Angstroms
zbox            length of c-axis of periodic box in Angstroms
alpha           angle between b- and c-axes of box in degrees
beta            angle between a- and c-axes of box in degrees
gamma           angle between a- and b-axes of box in degrees
xbox2           half of the a-axis length of periodic box
ybox2           half of the b-axis length of periodic box
zbox2           half of the c-axis length of periodic box
box34           three-fourths axis length of truncated octahedron
volbox          volume in Ang**3 of the periodic box
```

(continues on next page)

(continued from previous page)

beta_sin	sine of the beta periodic box angle
beta_cos	cosine of the beta periodic box angle
gamma_sin	sine of the gamma periodic box angle
gamma_cos	cosine of the gamma periodic box angle
beta_term	term used in generating triclinic box
gamma_term	term used in generating triclinic box
lvec	real space lattice vectors as matrix rows
recip	reciprocal lattice vectors as matrix columns
orthogonal	flag to mark periodic box as orthogonal
monoclinic	flag to mark periodic box as monoclinic
triclinic	flag to mark periodic box as triclinic
octahedron	flag to mark box as truncated octahedron
spacegrp	space group symbol for the unit cell type

CELL Module replicated cell periodic boundaries

ncell	total number of cell replicates for periodic boundaries
icell	offset along axes for each replicate periodic cell
xcell	length of the a-axis of the complete replicated cell
ycell	length of the b-axis of the complete replicated cell
zcell	length of the c-axis of the complete replicated cell
xcell2	half the length of the a-axis of the replicated cell
ycell2	half the length of the b-axis of the replicated cell
zcell2	half the length of the c-axis of the replicated cell

CHARGE Module partial charges in current structure

nion	total number of partial charges in system
iion	number of the atom site for each partial charge
jion	neighbor generation site for each partial charge
kion	cutoff switching site for each partial charge
pchg	magnitude of the partial charges (e-)

CHGPEN Module charge penetration in current structure

ncp	total number of charge penetration sites in system
pcore	number of core electrons at each multipole site
pval	number of valence electrons at each multipole site
palpha	charge penetration damping at each multipole site

CHGPOT Module charge-charge functional form details

electric	energy factor in kcal/mole for current force field
dielec	dielectric constant for electrostatic interactions
ebuffer	electrostatic buffering constant added to distance
c1scale	factor by which 1-1 charge interactions are scaled
c2scale	factor by which 1-2 charge interactions are scaled
c3scale	factor by which 1-3 charge interactions are scaled
c4scale	factor by which 1-4 charge interactions are scaled
c5scale	factor by which 1-5 charge interactions are scaled
neutnbr	logical flag governing use of neutral group neighbors
neutcut	logical flag governing use of neutral group cutoffs

CHGTRN Module charge transfer for current structure

```
nct             total number of dispersion sites in the system
chgct           charge for charge transfer at each multipole site
dmpct           charge transfer damping factor at each multipole site
```

CHRONO Module clock time values for current program

```
twall           current processor wall clock time in seconds
tcpu            elapsed cpu time from start of program in seconds
```

CHUNKS Module PME grid spatial decomposition values

```
nchunk          total number of spatial regions for PME grid
nchk1           number of spatial regions along the a-axis
nchk2           number of spatial regions along the b-axis
nchk3           number of spatial regions along the c-axis
ngrd1           number of grid points per region along a-axis
ngrd2           number of grid points per region along b-axis
ngrd3           number of grid points per region along c-axis
nlpts           PME grid points to the left of center point
nrpts           PME grid points to the right of center point
grdoff          offset for index into B-spline coefficients
pmetable        PME grid spatial regions involved for each site
```

COUPLE Module atom neighbor connectivity lists

```
n12             number of atoms directly bonded to each atom
n13             number of atoms in a 1-3 relation to each atom
n14             number of atoms in a 1-4 relation to each atom
n15             number of atoms in a 1-5 relation to each atom
i12             atom numbers of atoms 1-2 connected to each atom
i13             atom numbers of atoms 1-3 connected to each atom
i14             atom numbers of atoms 1-4 connected to each atom
i15             atom numbers of atoms 1-5 connected to each atom
```

CTRPOT Module charge transfer functional form details

```
ctrntyp         type of charge transfer term (SEPARATE or COMBINED)
```

DERIV Module Cartesian coord derivative components

```
desum           total energy Cartesian coordinate derivatives
deb             bond stretch Cartesian coordinate derivatives
dea             angle bend Cartesian coordinate derivatives
deba            stretch-bend Cartesian coordinate derivatives
deub            Urey-Bradley Cartesian coordinate derivatives
deaa            angle-angle Cartesian coordinate derivatives
deopb           out-of-plane bend Cartesian coordinate derivatives
deopd           out-of-plane distance Cartesian coordinate derivatives
deid            improper dihedral Cartesian coordinate derivatives
deit            improper torsion Cartesian coordinate derivatives
```

(continues on next page)

(continued from previous page)

det	torsional Cartesian coordinate derivatives
dept	pi-system torsion Cartesian coordinate derivatives
debt	stretch-torsion Cartesian coordinate derivatives
deat	angle-torsion Cartesian coordinate derivatives
dett	torsion-torsion Cartesian coordinate derivatives
dev	van der Waals Cartesian coordinate derivatives
der	Pauli repulsion Cartesian coordinate derivatives
dedsp	damped dispersion Cartesian coordinate derivatives
dec	charge-charge Cartesian coordinate derivatives
decd	charge-dipole Cartesian coordinate derivatives
ded	dipole-dipole Cartesian coordinate derivatives
dem	multipole Cartesian coordinate derivatives
dep	polarization Cartesian coordinate derivatives
dect	charge transfer Cartesian coordinate derivatives
derxf	reaction field Cartesian coordinate derivatives
des	solvation Cartesian coordinate derivatives
delf	metal ligand field Cartesian coordinate derivatives
deg	geometric restraint Cartesian coordinate derivatives
dex	extra energy term Cartesian coordinate derivatives

DIPOLE Module bond dipoles in current structure

ndipole	total number of dipoles in the system
idpl	numbers of atoms that define each dipole
bdpl	magnitude of each of the dipoles (Debye)
sdpl	position of each dipole between defining atoms

DISGEO Module distance geometry bounds & parameters

vdwmax	maximum value of hard sphere sum for an atom pair
compact	index of local distance compaction on embedding
pathmax	maximum value of upper bound after smoothing
dbnd	distance geometry upper and lower bounds matrix
georad	hard sphere radii for distance geometry atoms
use_invert	flag to use enantiomer closest to input structure
use_anneal	flag to use simulated annealing refinement

DISP Module damped dispersion for current structure

ndisp	total number of dispersion sites in the system
idisp	number of the atom for each dispersion site
csixpr	pairwise sum of C6 dispersion coefficients
csix	C6 dispersion coefficient value at each site
adisp	alpha dispersion damping value at each site

DMA Module QM spherical harmonic multipole moments

mp	atomic monopole charge values from DMA
dpx	atomic dipole moment x-component from DMA
dpy	atomic dipole moment y-component from DMA
dpz	atomic dipole moment z-component from DMA

(continues on next page)

(continued from previous page)

```
q20             atomic Q20 quadrupole component from DMA (zz)
q21c            atomic Q21c quadrupole component from DMA (xz)
q21s            atomic Q21s quadrupole component from DMA (yz)
q22c            atomic Q22c quadrupole component from DMA (xx-yy)
q22s            atomic Q22s quadrupole component from DMA (xy)
```

DOMEGA Module derivative components over torsions

```
tesum           total energy derivatives over torsions
teb             bond stretch derivatives over torsions
tea             angle bend derivatives over torsions
teba            stretch-bend derivatives over torsions
teub            Urey-Bradley derivatives over torsions
teaa            angle-angle derivatives over torsions
teopb           out-of-plane bend derivatives over torsions
teopd           out-of-plane distance derivatives over torsions
teid            improper dihedral derivatives over torsions
teit            improper torsion derivatives over torsions
tet             torsional derivatives over torsions
tept            pi-system torsion derivatives over torsions
tebt            stretch-torsion derivatives over torsions
teat            angle-torsion derivatives over torsions
tett            torsion-torsion derivatives over torsions
tev             van der Waals derivatives over torsions
ter             Pauli repulsion derivatives over torsions
tedsp           dampled dispersion derivatives over torsions
tec             charge-charge derivatives over torsions
tecd            charge-dipole derivatives over torsions
ted             dipole-dipole derivatives over torsions
tem             atomic multipole derivatives over torsions
tep             polarization derivatives over torsions
tect            charge transfer derivatives over torsions
terxf           reaction field derivatives over torsions
tes             solvation derivatives over torsions
telf            metal ligand field derivatives over torsions
teg             geometric restraint derivatives over torsions
tex             extra energy term derivatives over torsions
```

DSPPOT Module dispersion interaction scale factors

```
dsp2scale       scale factor for 1-2 dispersion energy interactions
dsp3scale       scale factor for 1-3 dispersion energy interactions
dsp4scale       scale factor for 1-4 dispersion energy interactions
dsp5scale       scale factor for 1-5 dispersion energy interactions
use_dcorr       flag to use long range dispersion correction
```

ENERGI Module individual potential energy components

```
esum            total potential energy of the system
eb              bond stretch potential energy of the system
ea              angle bend potential energy of the system
```

(continues on next page)

(continued from previous page)

```
eba             stretch-bend potential energy of the system
eub             Urey-Bradley potential energy of the system
eaa             angle-angle potential energy of the system
eopb            out-of-plane bend potential energy of the system
eopd            out-of-plane distance potential energy of the system
eid             improper dihedral potential energy of the system
eit             improper torsion potential energy of the system
et              torsional potential energy of the system
ept             pi-system torsion potential energy of the system
ebt             stretch-torsion potential energy of the system
eat             angle-torsion potential energy of the system
ett             torsion-torsion potential energy of the system
ev              van der Waals potential energy of the system
er              Pauli repulsion potential energy of the system
edsp            dampled dispersion potential energy of the system
ec              charge-charge potential energy of the system
ecd             charge-dipole potential energy of the system
ed              dipole-dipole potential energy of the system
em              atomic multipole potential energy of the system
ep              polarization potential energy of the system
ect             charge transfer potential energy of the system
erxf            reaction field potential energy of the system
es              solvation potential energy of the system
elf             metal ligand field potential energy of the system
eg              geometric restraint potential energy of the system
ex              extra term potential energy of the system
```

EWALD Module Ewald summation parameters and options

```
aewald          current value of Ewald convergence coefficient
aeewald         Ewald convergence coefficient for electrostatics
apewald         Ewald convergence coefficient for polarization
adewald         Ewald convergence coefficient for dispersion
boundary        Ewald boundary condition; none, tinfoil or vacuum
```

FACES Module Connolly area and volume variables

```
maxcls          maximum number of neighboring atom pairs
maxtt           maximum number of temporary tori
maxt            maximum number of total tori
maxp            maximum number of probe positions
maxv            maximum number of vertices
maxen           maximum number of concave edges
maxfn           maximum number of concave faces
maxc            maximum number of circles
maxeq           maximum number of convex edges
maxfs           maximum number of saddle faces
maxfq           maximum number of convex faces
maxcy           maximum number of cycles
mxcyeq          maximum number of convex edge cycles
mxfqcy          maximum number of convex face cycles
```

FFT Module Fast Fourier transform control values

```
maxprime        maximum number of prime factors of FFT dimension
iprime          prime factorization of each FFT dimension (FFTPACK)
planf           pointer to forward transform data structure (FFTW)
planb           pointer to backward transform data structure (FFTW)
ffttable        intermediate array used by the FFT routine (FFTPACK)
ffttyp          type of FFT package; currently FFTPACK or FFTW
```

FIELDS Module molecular mechanics force field type

```
maxbio          maximum number of biopolymer atom definitions
biotyp          force field atom type of each biopolymer type
forcefield      string used to describe the current forcefield
```

FILES Module name & number of current structure file

```
nprior          number of previously existing cycle files
ldir            length in characters of the directory name
leng            length in characters of the base filename
filename        base filename used by default for all files
outfile         output filename used for intermediate results
```

FRACS Module distances to molecular center of mass

```
xfrac           fractional coordinate along a-axis of center of mass
yfrac           fractional coordinate along b-axis of center of mass
zfrac           fractional coordinate along c-axis of center of mass
```

FREEZE Module definition of holonomic constraints

```
nrat            number of holonomic distance constraints to apply
nratx           number of atom group holonomic constraints to apply
iratx           group number of group in a holonomic constraint
kratx           spatial constraint type (1=plane, 2=line, 3=point)
irat            atom numbers of atoms in a holonomic constraint
rateps          convergence tolerance for holonomic constraints
krat            ideal distance value for holonomic constraint
use_rattle      logical flag to set use of holonomic contraints
ratimage        flag to use minimum image for holonomic constraint
```

GKSTUF Module generalized Kirkwood solvation values

```
gkc             tuning parameter exponent in the f(GB) function
gkr             generalized Kirkwood cavity radii for atom types
```

GROUP Module partitioning of system into atom groups

```
ngrp            total number of atom groups in the system
kgrp            contiguous list of the atoms in each group
grplist         number of the group to which each atom belongs
igrp            first and last atom of each group in the list
```

(continues on next page)

(continued from previous page)

grpmass	total mass of all the atoms in each group
wgrp	weight for each set of group-group interactions
use_group	flag to use partitioning of system into groups
use_intra	flag to include only intragroup interactions
use_inter	flag to include only intergroup interactions

HESCUT Module cutoff for Hessian matrix elements

hesscut	magnitude of smallest allowed Hessian element

HESSN Module Cartesian Hessian elements for one atom

hessx	Hessian elements for x-component of current atom
hessy	Hessian elements for y-component of current atom
hessz	Hessian elements for z-component of current atom

HPMF Module hydrophobic potential of mean force term

rcarbon	radius of a carbon atom for use with HPMF
rwater	radius of a water molecule for use with HPMF
acsurf	surface area of a hydrophobic carbon atom
safact	constant for calculation of atomic surface area
tgrad	tanh slope (set very steep, default=100)
toffset	shift the tanh plot along the x-axis (default=6)
hpmfcut	cutoff distance for pairwise HPMF interactions
hd1	hd2,hd3 hydrophobic PMF well depth parameter
hc1	hc2,hc3 hydrophobic PMF well center point
hw1	hw2,hw3 reciprocal of the hydrophobic PMF well width
npmf	number of hydrophobic carbon atoms in the system
ipmf	number of the atom for each HPMF carbon atom site
rpmf	radius of each atom for use with hydrophobic PMF
acsa	SASA value for each hydrophobic PMF carbon atom

IELSCF Module extended Lagrangian induced dipoles

nfree_aux	total degrees of freedom for auxiliary dipoles
tautemp_aux	time constant for aululiary Berendsen thermostat
kelvin_aux	target system temperature for auxiliary dipoles
uaux	auxiliary induced dipole value at each site
upaux	auxiliary shadow induced dipoles at each site
vaux	auxiliary induced dipole velocity at each site
vpaux	auxiliary shadow dipole velocity at each site
aaux	auxiliary induced dipole acceleration at each site
apaux	auxiliary shadow dipole acceleration at each site
use_ielscf	flag to use inertial extended Lagrangian method

IMPROP Module improper dihedrals in current structure

niprop	total number of improper dihedral angles in the system
iiprop	numbers of the atoms in each improper dihedral angle
kprop	force constant values for improper dihedral angles
vprop	ideal improper dihedral angle value in degrees

IMPTOR Module improper torsions in current structure

```
nitors          total number of improper torsional angles in the system
iitors          numbers of the atoms in each improper torsional angle
itors1          1-fold amplitude and phase for each improper torsion
itors2          2-fold amplitude and phase for each improper torsion
itors3          3-fold amplitude and phase for each improper torsion
```

INFORM Module program I/O and flow control values

```
maxask          maximum number of queries for interactive input
digits          decimal places output for energy and coordinates
iprint          steps between status printing (0=no printing)
iwrite          steps between coordinate saves (0=no saves)
isend           steps between socket communication (0=no sockets)
silent          logical flag to turn off all information printing
verbose         logical flag to turn on extra information printing
debug           logical flag to turn on full debug printing
holdup          logical flag to wait for carriage return on exit
abort           logical flag to stop execution at next chance
```

INTER Module sum of intermolecular energy components

```
einter          total intermolecular potential energy
```

IOUNIT Module Fortran input/output unit numbers

```
input           Fortran I/O unit for main input (default=5)
iout            Fortran I/O unit for main output (default=6)
```

KANANG Module angle-angle term forcefield parameters

```
anan            angle-angle cross term parameters for each atom class
```

KANGS Module bond angle bend forcefield parameters

```
maxna           maximum number of harmonic angle bend parameter entries
maxna5          maximum number of 5-membered ring angle bend entries
maxna4          maximum number of 4-membered ring angle bend entries
maxna3          maximum number of 3-membered ring angle bend entries
maxnap          maximum number of in-plane angle bend parameter entries
maxnaf          maximum number of Fourier angle bend parameter entries
acon            force constant parameters for harmonic angle bends
acon5           force constant parameters for 5-ring angle bends
acon4           force constant parameters for 4-ring angle bends
acon3           force constant parameters for 3-ring angle bends
aconp           force constant parameters for in-plane angle bends
aconf           force constant parameters for Fourier angle bends
ang             bond angle parameters for harmonic angle bends
ang5            bond angle parameters for 5-ring angle bends
ang4            bond angle parameters for 4-ring angle bends
ang3            bond angle parameters for 3-ring angle bends
```

(continues on next page)

(continued from previous page)

angp	bond angle parameters for in-plane angle bends
angf	phase shift angle and periodicity for Fourier bends
ka	string of atom classes for harmonic angle bends
ka5	string of atom classes for 5-ring angle bends
ka4	string of atom classes for 4-ring angle bends
ka3	string of atom classes for 3-ring angle bends
kap	string of atom classes for in-plane angle bends
kaf	string of atom classes for Fourier angle bends

KANTOR Module angle-torsion forcefield parameters

maxnat	maximum number of angle-torsion parameter entries
atcon	torsional amplitude parameters for angle-torsion
kat	string of atom classes for angle-torsion terms

KATOMS Module atom definition forcefield parameters

atmcls	atom class number for each of the atom types
atmnum	atomic number for each of the atom types
ligand	number of atoms to be attached to each atom type
weight	average atomic mass of each atom type
symbol	modified atomic symbol for each atom type
describe	string identifying each of the atom types

KBONDS Module bond stretching forcefield parameters

maxnb	maximum number of bond stretch parameter entries
maxnb5	maximum number of 5-membered ring bond stretch entries
maxnb4	maximum number of 4-membered ring bond stretch entries
maxnb3	maximum number of 3-membered ring bond stretch entries
maxnel	maximum number of electronegativity bond corrections
bcon	force constant parameters for harmonic bond stretch
bcon5	force constant parameters for 5-ring bond stretch
bcon4	force constant parameters for 4-ring bond stretch
bcon3	force constant parameters for 3-ring bond stretch
blen	bond length parameters for harmonic bond stretch
blen5	bond length parameters for 5-ring bond stretch
blen4	bond length parameters for 4-ring bond stretch
blen3	bond length parameters for 3-ring bond stretch
dlen	electronegativity bond length correction parameters
kb	string of atom classes for harmonic bond stretch
kb5	string of atom classes for 5-ring bond stretch
kb4	string of atom classes for 4-ring bond stretch
kb3	string of atom classes for 3-ring bond stretch
kel	string of atom classes for electronegativity corrections

KCHRGE Module partial charge forcefield parameters

chg	partial charge parameters for each atom type

KCPEN Module charge penetration forcefield parameters

```
cpele           valence electron magnitude for each atom class
cpalp           alpha charge penetration parameter for each atom class
```

KCTRN Module charge transfer forcefield parameters

```
ctchg           charge transfer magnitude for each atom class
ctdmp           alpha charge transfer parameter for each atom class
```

KDIPOL Module bond dipole forcefield parameters

```
maxnd           maximum number of bond dipole parameter entries
maxnd5          maximum number of 5-membered ring dipole entries
maxnd4          maximum number of 4-membered ring dipole entries
maxnd3          maximum number of 3-membered ring dipole entries
dpl             dipole moment parameters for bond dipoles
dpl5            dipole moment parameters for 5-ring dipoles
dpl4            dipole moment parameters for 4-ring dipoles
dpl3            dipole moment parameters for 3-ring dipoles
pos             dipole position parameters for bond dipoles
pos5            dipole position parameters for 5-ring dipoles
pos4            dipole position parameters for 4-ring dipoles
pos3            dipole position parameters for 3-ring dipoles
kd              string of atom classes for bond dipoles
kd5             string of atom classes for 5-ring dipoles
kd4             string of atom classes for 4-ring dipoles
kd3             string of atom classes for 3-ring dipoles
```

KDSP Module damped dispersion forcefield parameters

```
dspsix          C6 dispersion coefficient for each atom class
dspdmp          alpha dispersion parameter for each atom class
```

KEYS Module contents of the keyword control file

```
maxkey          maximum number of lines in the keyword file
nkey            number of nonblank lines in the keyword file
keyline         contents of each individual keyword file line
```

KHBOND Module H-bonding term forcefield parameters

```
maxnhb          maximum number of hydrogen bonding pair entries
radhb           radius parameter for hydrogen bonding pairs
epshb           well depth parameter for hydrogen bonding pairs
khb             string of atom types for hydrogen bonding pairs
```

KIPROP Module improper dihedral forcefield parameters

```
maxndi          maximum number of improper dihedral parameter entries
dcon            force constant parameters for improper dihedrals
tdi             ideal dihedral angle values for improper dihedrals
kdi             string of atom classes for improper dihedral angles
```

KITORS Module improper torsion forcefield parameters

```
maxnti          maximum number of improper torsion parameter entries
ti1             torsional parameters for improper 1-fold rotation
ti2             torsional parameters for improper 2-fold rotation
ti3             torsional parameters for improper 3-fold rotation
kti             string of atom classes for improper torsional parameters
```

KMULTI Module atomic multipole forcefield parameters

```
maxnmp          maximum number of atomic multipole parameter entries
multip          atomic monopole, dipole and quadrupole values
mpaxis          type of local axis definition for atomic multipoles
kmp             string of atom types for atomic multipoles
```

KOPBND Module out-of-plane bend forcefield parameters

```
maxnopb         maximum number of out-of-plane bending entries
opbn            force constant parameters for out-of-plane bending
kopb            string of atom classes for out-of-plane bending
```

KOPDST Module out-of-plane distance forcefield params

```
maxnopd         maximum number of out-of-plane distance entries
opds            force constant parameters for out-of-plane distance
kopd            string of atom classes for out-of-plane distance
```

KORBS Module pisystem orbital forcefield parameters

```
maxnpi          maximum number of pisystem bond parameter entries
maxnpi5         maximum number of 5-membered ring pibond entries
maxnpi4         maximum number of 4-membered ring pibond entries
sslope          slope for bond stretch vs. pi-bond order
sslope5         slope for 5-ring bond stretch vs. pi-bond order
sslope4         slope for 4-ring bond stretch vs. pi-bond order
tslope          slope for 2-fold torsion vs. pi-bond order
tslope5         slope for 5-ring 2-fold torsion vs. pi-bond order
tslope4         slope for 4-ring 2-fold torsion vs. pi-bond order
electron        number of pi-electrons for each atom class
ionize          ionization potential for each atom class
repulse         repulsion integral value for each atom class
kpi             string of atom classes for pisystem bonds
kpi5            string of atom classes for 5-ring pisystem bonds
kpi4            string of atom classes for 4-ring pisystem bonds
```

KPITOR Module pi-system torsion forcefield parameters

```
maxnpt          maximum number of pi-system torsion parameter entries
ptcon           force constant parameters for pi-system torsions
kpt             string of atom classes for pi-system torsion terms
```

KPOLR Module polarizability forcefield parameters

```
pgrp            connected types in polarization group of each atom type
polr            dipole polarizability parameters for each atom type
athl            Thole polarizability damping value for each atom type
ddir            direct polarization damping value for each atom type
```

KREPL Module Pauli repulsion forcefield parameters

```
prsiz           Pauli repulsion size value for each atom class
prdmp           alpha Pauli repulsion parameter for each atom class
prele           number of valence electrons for each atom class
```

KSTBND Module stretch-bend forcefield parameters

```
maxnsb          maximum number of stretch-bend parameter entries
stbn            force constant parameters for stretch-bend terms
ksb             string of atom classes for stretch-bend terms
```

KSTTOR Module stretch-torsion forcefield parameters

```
maxnbt          maximum number of stretch-torsion parameter entries
btcon           torsional amplitude parameters for stretch-torsion
kbt             string of atom classes for stretch-torsion terms
```

KTORSN Module torsional angle forcefield parameters

```
maxnt           maximum number of torsional angle parameter entries
maxnt5          maximum number of 5-membered ring torsion entries
maxnt4          maximum number of 4-membered ring torsion entries
t1              torsional parameters for standard 1-fold rotation
t2              torsional parameters for standard 2-fold rotation
t3              torsional parameters for standard 3-fold rotation
t4              torsional parameters for standard 4-fold rotation
t5              torsional parameters for standard 5-fold rotation
t6              torsional parameters for standard 6-fold rotation
t15             torsional parameters for 1-fold rotation in 5-ring
t25             torsional parameters for 2-fold rotation in 5-ring
t35             torsional parameters for 3-fold rotation in 5-ring
t45             torsional parameters for 4-fold rotation in 5-ring
t55             torsional parameters for 5-fold rotation in 5-ring
t65             torsional parameters for 6-fold rotation in 5-ring
t14             torsional parameters for 1-fold rotation in 4-ring
t24             torsional parameters for 2-fold rotation in 4-ring
t34             torsional parameters for 3-fold rotation in 4-ring
t44             torsional parameters for 4-fold rotation in 4-ring
t54             torsional parameters for 5-fold rotation in 4-ring
t64             torsional parameters for 6-fold rotation in 4-ring
kt              string of atom classes for torsional angles
kt5             string of atom classes for 5-ring torsions
kt4             string of atom classes for 4-ring torsions
```

KTRTOR Module torsion-torsion forcefield parameters

maxntt	maximum number of torsion-torsion parameter entries
maxtgrd	maximum dimension of torsion-torsion spline grid
maxtgrd2	maximum number of torsion-torsion spline grid points
tnx	number of columns in torsion-torsion spline grid
tny	number of rows in torsion-torsion spline grid
ttx	angle values for first torsion of spline grid
tty	angle values for second torsion of spline grid
tbf	function values at points on spline grid
tbx	gradient over first torsion of spline grid
tby	gradient over second torsion of spline grid
tbxy	Hessian cross components over spline grid
ktt	string of torsion-torsion atom classes

KURYBR Module Urey-Bradley term forcefield parameters

maxnu	maximum number of Urey-Bradley parameter entries
ucon	force constant parameters for Urey-Bradley terms
dst13	ideal 1-3 distance parameters for Urey-Bradley terms
ku	string of atom classes for Urey-Bradley terms

KVDWPR Module special vdw term forcefield parameters

maxnvp	maximum number of special van der Waals pair entries
radpr	radius parameter for special van der Waals pairs
epspr	well depth parameter for special van der Waals pairs
kvpr	string of atom classes for special van der Waals pairs

KVDWS Module van der Waals term forcefield parameters

rad	van der Waals radius parameter for each atom type
eps	van der Waals well depth parameter for each atom type
rad4	van der Waals radius parameter in 1-4 interactions
eps4	van der Waals well depth parameter in 1-4 interactions
reduct	van der Waals reduction factor for each atom type

LIGHT Module method of lights pair neighbors indices

nlight	total number of sites for method of lights calculation
kbx	low index of neighbors of each site in the x-sorted list
kby	low index of neighbors of each site in the y-sorted list
kbz	low index of neighbors of each site in the z-sorted list
kex	high index of neighbors of each site in the x-sorted list
key	high index of neighbors of each site in the y-sorted list
kez	high index of neighbors of each site in the z-sorted list
locx	maps the x-sorted list into original interaction list
locy	maps the y-sorted list into original interaction list
locz	maps the z-sorted list into original interaction list
rgx	maps the original interaction list into x-sorted list
rgy	maps the original interaction list into y-sorted list
rgz	maps the original interaction list into z-sorted list

LIMITS Module interaction taper & cutoff distances

```
vdwcut          cutoff distance for van der Waals interactions
repcut          cutoff distance for Pauli repulsion interactions
dispcut         cutoff distance for dispersion interactions
chgcut          cutoff distance for charge-charge interactions
dplcut          cutoff distance for dipole-dipole interactions
mpolecut        cutoff distance for atomic multipole interactions
ctrncut         cutoff distance for charge transfer interactions
vdwtaper        distance at which van der Waals switching begins
reptaper        distance at which Pauli repulsion switching begins
disptaper       distance at which dispersion switching begins
chgtaper        distance at which charge-charge switching begins
dpltaper        distance at which dipole-dipole switching begins
mpoletaper      distance at which atomic multipole switching begins
ctrntaper       distance at which charge transfer switching begins
ewaldcut        cutoff distance for real space Ewald electrostatics
dewaldcut       cutoff distance for real space Ewald dispersion
usolvcut        cutoff distance for dipole solver preconditioner
use_ewald       logical flag governing use of electrostatic Ewald
use_dewald      logical flag governing use of dispersion Ewald
use_lights      logical flag governing use of method of lights
use_list        logical flag governing use of any neighbor lists
use_vlist       logical flag governing use of van der Waals list
use_dlist       logical flag governing use of dispersion list
use_clist       logical flag governing use of charge list
use_mlist       logical flag governing use of multipole list
use_ulist       logical flag governing use of preconditioner list
```

LINMIN Module line search minimization parameters

```
stpmin          minimum step length in current line search direction
stpmax          maximum step length in current line search direction
cappa           stringency of line search (0=tight < cappa < 1=loose)
slpmax          projected gradient above which stepsize is reduced
angmax          maximum angle between search direction and -gradient
intmax          maximum number of interpolations during line search
```

MATH Module mathematical and geometrical constants

```
pi              numerical value of the geometric constant
elog            numerical value of the natural logarithm base
radian          conversion factor from radians to degrees
logten          numerical value of the natural log of ten
twosix          numerical value of the sixth root of two
sqrtpi          numerical value of the square root of Pi
sqrttwo         numerical value of the square root of two
sqrtthree       numerical value of the square root of three
```

MDSTUF Module molecular dynamics trajectory controls

```
nfree           total number of degrees of freedom for a system
irest           steps between removal of COM motion (0=no removal)
bmnmix          mixing coefficient for use with Beeman integrator
```

(continues on next page)

(continued from previous page)

```
arespa          inner time step for use with RESPA integrator
dorest          logical flag to remove center of mass motion
integrate       type of molecular dynamics integration algorithm
```

MERCK Module MMFF-specific force field parameters

```
nlignes         number of atom pairs having MMFF Bond Type 1
bt_1            atom pairs having MMFF Bond Type 1
eqclass         table of atom class equivalencies used to find
default         parameters if explicit values are missing
see             J. Comput. Chem., 17, 490-519, '95, Table IV)
crd             number of attached neighbors    |
val             valency value                   |  see T. A. Halgren,
pilp            if 0, no lone pair              |  J. Comput. Chem.,
if              1, one or more lone pair(s)  |  17, 616-645 (1995)
mltb            multibond indicator             |
arom            aromaticity indicator           |
lin             linearity indicator             |
sbmb            single- vs multiple-bond flag   |
mmffarom        aromatic rings parameters
mmffaromc       cationic aromatic rings parameters
mmffaroma       anionic aromatic rings parameters
```

MINIMA Module general parameters for minimizations

```
fctmin          value below which function is deemed optimized
hguess          initial value for the H-matrix diagonal elements
maxiter         maximum number of iterations during optimization
nextiter        iteration number to use for the first iteration
```

MOLCUL Module individual molecules in current system

```
nmol            total number of separate molecules in the system
imol            first and last atom of each molecule in the list
kmol            contiguous list of the atoms in each molecule
molcule         number of the molecule to which each atom belongs
totmass         total weight of all the molecules in the system
molmass         molecular weight for each molecule in the system
```

MOLDYN Module MD trajectory velocity & acceleration

```
v               current velocity of each atom along the x,y,z-axes
a               current acceleration of each atom along x,y,z-axes
aalt            alternate acceleration of each atom along x,y,z-axes
```

MOMENT Module electric multipole moment components

```
netchg          net electric charge for the total system
netdpl          dipole moment magnitude for the total system
netqdp          diagonal quadrupole (Qxx, Qyy, Qzz) for system
xdpl            dipole vector x-component in the global frame
```

(continues on next page)

(continued from previous page)

```
ydpl            dipole vector y-component in the global frame
zdpl            dipole vector z-component in the global frame
xxqdp           quadrupole tensor xx-component in global frame
xyqdp           quadrupole tensor xy-component in global frame
xzqdp           quadrupole tensor xz-component in global frame
yxqdp           quadrupole tensor yx-component in global frame
yyqdp           quadrupole tensor yy-component in global frame
yzqdp           quadrupole tensor yz-component in global frame
zxqdp           quadrupole tensor zx-component in global frame
zyqdp           quadrupole tensor zy-component in global frame
zzqdp           quadrupole tensor zz-component in global frame
```

MPLPOT Module multipole functional form details

```
m2scale         scale factor for 1-2 multipole energy interactions
m3scale         scale factor for 1-3 multipole energy interactions
m4scale         scale factor for 1-4 multipole energy interactions
m5scale         scale factor for 1-5 multipole energy interactions
use_chgpen      flag to use charge penetration damped potential
pentyp          type of penetration damping (NONE, GORDON1, GORDON2)
```

MPOLE Module atomic multipoles in current structure

```
maxpole         max components (monopole=1,dipole=4,quadrupole=13)
npole           total number of multipole sites in the system
ipole           number of the atom for each multipole site
polsiz          number of multipole components at each atom
pollist         multipole site for each atom (0=no multipole)
zaxis           number of the z-axis defining atom for each site
xaxis           number of the x-axis defining atom for each site
yaxis           number of the y-axis defining atom for each site
pole            traceless Cartesian multipoles in the local frame
rpole           traceless Cartesian multipoles in the global frame
spole           spherical harmonic multipoles in the local frame
srpole          spherical harmonic multipoles in the global frame
polaxe          local axis type for each multipole site
```

MRECIP Module reciprocal PME for permanent multipoles

```
vmxx            scalar sum xx-component of virial due to multipoles
vmyy            scalar sum yy-component of virial due to multipoles
vmzz            scalar sum zz-component of virial due to multipoles
vmxy            scalar sum xy-component of virial due to multipoles
vmxz            scalar sum xz-component of virial due to multipoles
vmyz            scalar sum yz-component of virial due to multipoles
cmp             Cartesian permenent multipoles as polytensor vector
fmp             fractional permanent multipoles as polytensor vector
cphi            Cartesian permanent multipole potential and field
fphi            fractional permanent multipole potential and field
```

MUTANT Module free energy calculation hybrid atoms

```
nmut            number of atoms mutated from initial to final state
vcouple         van der Waals lambda type (0=decouple, 1=annihilate)
imut            atomic sites differing in initial and final state
type0           atom type of each atom in the initial state system
class0          atom class of each atom in the initial state system
type1           atom type of each atom in the final state system
class1          atom class of each atom in the final state system
lambda          generic weighting between initial and final states
tlambda         state weighting value for torsional potential
vlambda         state weighting value for van der Waals potentials
elambda         state weighting value for electrostatic potentials
scexp           scale factor for soft core buffered 14-7 potential
scalpha         scale factor for soft core buffered 14-7 potential
mut             true if an atom is to be mutated, false otherwise
```

NEIGH Module pairwise neighbor list indices & storage

```
maxvlst         maximum size of van der Waals pair neighbor lists
maxelst         maximum size of electrostatic pair neighbor lists
maxulst         maximum size of dipole preconditioner pair lists
nvlst           number of sites in list for each vdw site
vlst            site numbers in neighbor list of each vdw site
nelst           number of sites in list for each electrostatic site
elst            site numbers in list of each electrostatic site
nulst           number of sites in list for each preconditioner site
ulst            site numbers in list of each preconditioner site
lbuffer         width of the neighbor list buffer region
pbuffer         width of the preconditioner list buffer region
lbuf2           square of half the neighbor list buffer width
pbuf2           square of half the preconditioner list buffer width
vbuf2           square of van der Waals cutoff plus the list buffer
vbufx           square of van der Waals cutoff plus 2X list buffer
dbuf2           square of dispersion cutoff plus the list buffer
dbufx           square of dispersion cutoff plus 2X list buffer
cbuf2           square of charge cutoff plus the list buffer
cbufx           square of charge cutoff plus 2X list buffer
mbuf2           square of multipole cutoff plus the list buffer
mbufx           square of multipole cutoff plus 2X list buffer
ubuf2           square of preconditioner cutoff plus the list buffer
ubufx           square of preconditioner cutoff plus 2X list buffer
xvold           x-coordinate at last vdw/dispersion list update
yvold           y-coordinate at last vdw/dispersion list update
zvold           z-coordinate at last vdw/dispersion list update
xeold           x-coordinate at last electrostatic list update
yeold           y-coordinate at last electrostatic list update
zeold           z-coordinate at last electrostatic list update
xuold           x-coordinate at last preconditioner list update
yuold           y-coordinate at last preconditioner list update
zuold           z-coordinate at last preconditioner list update
dovlst          logical flag to rebuild vdw neighbor list
dodlst          logical flag to rebuild dispersion neighbor list
doclst          logical flag to rebuild charge neighbor list
```

(continues on next page)

(continued from previous page)

```
domlst          logical flag to rebuild multipole neighbor list
doulst          logical flag to rebuild preconditioner neighbor list
```

NONPOL Module nonpolar cavity & dispersion parameters

```
epso            water oxygen eps for implicit dispersion term
epsh            water hydrogen eps for implicit dispersion term
rmino           water oxygen Rmin for implicit dispersion term
rminh           water hydrogen Rmin for implicit dispersion term
awater          water number density at standard temp & pressure
slevy           enthalpy-to-free energy scale factor for dispersion
solvprs         limiting microscopic solvent pressure value
surften         limiting macroscopic surface tension value
spcut           starting radius for solvent pressure tapering
spoff           cutoff radius for solvent pressure tapering
stcut           starting radius for surface tension tapering
stoff           cutoff radius for surface tension tapering
rcav            atomic radius of each atom for cavitation energy
rdisp           atomic radius of each atom for dispersion energy
cdisp           maximum dispersion energy for each atom
```

NUCLEO Module parameters for nucleic acid structure

```
pucker          sugar pucker, either 2=2'-endo or 3=3'-endo
glyco           glycosidic torsional angle for each nucleotide
bkbone          phosphate backbone angles for each nucleotide
dblhlx          flag to mark system as nucleic acid double helix
deoxy           flag to mark deoxyribose or ribose sugar units
hlxform         helix form (A, B or Z) of polynucleotide strands
```

OMEGA Module torsional space dihedral angle values

```
nomega          number of dihedral angles allowed to rotate
iomega          numbers of two atoms defining rotation axis
zline           line number in Z-matrix of each dihedral angle
dihed           current value in radians of each dihedral angle
```

OPBEND Module out-of-plane bends in current structure

```
nopbend         total number of out-of-plane bends in the system
iopb            bond angle numbers used in out-of-plane bending
opbk            force constant values for out-of-plane bending
```

OPDIST Module out-of-plane distances in structure

```
nopdist         total number of out-of-plane distances in the system
iopd            numbers of the atoms in each out-of-plane distance
opdk            force constant values for out-of-plane distance
```

OPENMP Module OpenMP processor and thread values

```
nproc           number of processors available to OpenMP
nthread         number of threads to be used with OpenMP
```

ORBITS Module conjugated pisystem orbital energies

```
qorb            number of pi-electrons contributed by each atom
worb            ionization potential of each pisystem atom
emorb           repulsion integral for each pisystem atom
```

OUTPUT Module output file format control parameters

```
archive         logical flag to save structures in an archive
noversion       logical flag governing use of filename versions
overwrite       logical flag to overwrite intermediate files inplace
cyclesave       logical flag to mark use of numbered cycle files
velsave         logical flag to save velocity vector components
frcsave         logical flag to save force vector components
uindsave        logical flag to save induced atomic dipoles
coordtype       selects Cartesian, internal, rigid body or none
```

PARAMS Module force field parameter file contents

```
maxprm          maximum number of lines in the parameter file
nprm            number of nonblank lines in the parameter file
prmline         contents of each individual parameter file line
```

PATHS Module Elber reaction path method parameters

```
pnorm           length of the reactant-product vector
acoeff          transformation matrix 'A' from Elber algorithm
pc0             reactant Cartesian coordinates as variables
pc1             product Cartesian coordinates as variables
pvect           vector connecting the reactant and product
pstep           step per cycle along reactant-product vector
pzet            current projection on reactant-product vector
gc              gradient of the path constraints
```

PBSTUF Module Poisson-Boltzmann solvation parameters

```
APBS            configuration parameters (see APBS documentation for details)
In              the column on the right are possible values for each variable,
with            default values given in brackets. Only a subset of the APBS
options         are supported and/or are appropriate for use with AMOEBA
pbtyp           lpbe
At              some point AMOEBA with the non-linear PBE could be supported,
but             this is only worked out for energies (no gradients)
pbsoln          mg-auto, [mg-manual]
Currently       there is only limited support for focusing calculations,
which           is a powerful feature of APBS. At present, all energies and
forces          must all be calculated using the finest solution
bcfl            boundary conditions             zero, sdh, [mdh]
chgm            multipole discretization        spl4
```

(continues on next page)

(continued from previous page)

```
other           charge discretization methods are not appropriate for AMOEBA
srfm            surface method                  mol, smol, [spl4]
spl4            is required for forces calculations, although mol is useful
for             comparison with generalized Kirkwood
dime            number of grid points           [65, 65, 65]
grid            grid spacing (mg-manual)        fxn of "dime"
cgrid           coarse grid spacing             fxn of "dime"
fgrid           fine grid spacing               cgrid / 2
stable          results require grid spacing to be fine enough to keep
multipoles      inside the dielectric boundary (2.5 * grid < PBR)
gcent           grid center (mg-manual)         center of mass
cgcent          coarse grid center              center of mass
fgcent          fine grid center                center of mass
pdie            solute/homogeneous dieletric    [1.0]
sdie            solvent dieletric               [78.3]
ionn            number of ion species           [0]
ionc            ion concentration (M)           [0.0]
ionq            ion charge (electrons)          [1.0]
ionr            ion radius (A)                  [2.0]
srad            solvent probe radius (A)        [1.4]
swin            surface spline window width     [0.3]
sdens           density of surface points       [10.0]
additional      parameter to facilitate default grid setup
smin            minimum distance between an     [10.0]
atom            and the grid boundary (A)
pbe             Poisson-Boltzmann permanent multipole solvation energy
apbe            Poisson-Boltzmann permanent multipole energy over atoms
pbr             Poisson-Boltzmann cavity radii for atom types
pbep            Poisson-Boltzmann energies on permanent multipoles
pbfp            Poisson-Boltzmann forces on permanent multipoles
pbtp            Poisson-Boltzmann torques on permanent multipoles
pbeuind         Poisson-Boltzmann field due to induced dipoles
pbeuinp         Poisson-Boltzmann field due to non-local induced dipoles
```

PDB Module Protein Data Bank structure definition

```
npdb            number of atoms stored in Protein Data Bank format
nres            number of residues stored in Protein Data Bank format
resnum          number of the residue to which each atom belongs
resatm          number of first and last atom in each residue
npdb12          number of atoms directly bonded to each CONECT atom
ipdb12          atom numbers of atoms connected to each CONECT atom
pdblist         list of the Protein Data Bank atom number of each atom
xpdb            x-coordinate of each atom stored in PDB format
ypdb            y-coordinate of each atom stored in PDB format
zpdb            z-coordinate of each atom stored in PDB format
altsym          string with PDB alternate locations to be included
pdbres          Protein Data Bank residue name assigned to each atom
pdbatm          Protein Data Bank atom name assigned to each atom
pdbtyp          Protein Data Bank record type assigned to each atom
chnsym          string with PDB chain identifiers to be included
instyp          string with PDB insertion records to be included
```

PHIPSI Module phi-psi-omega-chi angles for protein

```
chiral          chirality of each amino acid residue (1=L, -1=D)
disulf          residue joined to each residue via a disulfide link
phi             value of the phi angle for each amino acid residue
psi             value of the psi angle for each amino acid residue
omg             value of the omega angle for each amino acid residue
chi             values of the chi angles for each amino acid residue
```

PIORBS Module conjugated system in current structure

```
norbit          total number of pisystem orbitals in the system
nconj           total number of separate conjugated piystems
reorbit         number of evaluations between orbital updates
nbpi            total number of bonds affected by the pisystem
ntpi            total number of torsions affected by the pisystem
iorbit          numbers of the atoms containing pisystem orbitals
iconj           first and last atom of each pisystem in the list
kconj           contiguous list of atoms in each pisystem
piperp          atoms defining a normal plane to each orbital
ibpi            bond and piatom numbers for each pisystem bond
itpi            torsion and pibond numbers for each pisystem torsion
pbpl            pi-bond orders for bonds in "planar" pisystem
pnpl            pi-bond orders for bonds in "nonplanar" pisystem
listpi          atom list indicating whether each atom has an orbital
```

PISTUF Module bond order-related pisystem parameters

```
bkpi            bond stretch force constants for pi-bond order of 1.0
blpi            ideal bond length values for a pi-bond order of 1.0
kslope          rate of force constant decrease with bond order decrease
lslope          rate of bond length increase with a bond order decrease
torsp2          2-fold torsional energy barrier for pi-bond order of 1.0
```

PITORS Module pi-system torsions in current structure

```
npitors         total number of pi-system torsional interactions
ipit            numbers of the atoms in each pi-system torsion
kpit            2-fold pi-system torsional force constants
```

PME Module values for particle mesh Ewald summation

```
nfft1           current number of PME grid points along a-axis
nfft2           current number of PME grid points along b-axis
nfft3           current number of PME grid points along c-axis
nefft1          number of grid points along electrostatic a-axis
nefft2          number of grid points along electrostatic b-axis
nefft3          number of grid points along electrostatic c-axis
ndfft1          number of grid points along dispersion a-axis
ndfft2          number of grid points along dispersion b-axis
ndfft3          number of grid points along dispersion c-axis
bsorder         current order of the PME B-spline values
bseorder        order of the electrostatic PME B-spline values
```

(continues on next page)

(continued from previous page)

```
bsporder        order of the polarization PME B-spline values
bsdorder        order of the dispersion PME B-spline values
igrid           initial Ewald grid values for B-spline
bsmod1          B-spline moduli along the a-axis direction
bsmod2          B-spline moduli along the b-axis direction
bsmod3          B-spline moduli along the c-axis direction
bsbuild         B-spline derivative coefficient temporary storage
thetai1         B-spline coefficients along the a-axis
thetai2         B-spline coefficients along the b-axis
thetai3         B-spline coefficients along the c-axis
qgrid           values on the particle mesh Ewald grid
qfac            prefactors for the particle mesh Ewald grid
```

POLAR Module induced dipole moments & polarizability

```
npolar          total number of polarizable sites in the system
ipolar          number of the multipole for each polarizable site
polarity        dipole polarizability for each multipole site (Ang**3)
thole           Thole polarizability damping value for each site
dirdamp         direct polarization damping value for each site
pdamp           value of polarizability scale factor for each site
udir            direct induced dipole components at each multipole site
udirp           direct induced dipoles in field used for energy terms
udirs           direct GK or PB induced dipoles at each multipole site
udirps          direct induced dipoles in field used for GK or PB energy
uind            mutual induced dipole components at each multipole site
uinp            mutual induced dipoles in field used for energy terms
uinds           mutual GK or PB induced dipoles at each multipole site
uinps           mutual induced dipoles in field used for GK or PB energy
uexact          exact SCF induced dipoles to full numerical precision
douind          flag to allow induced dipoles at each atomic site
```

POLGRP Module polarization group connectivity lists

```
maxp11          maximum number of atoms in a polarization group
maxp12          maximum number of atoms in groups 1-2 to an atom
maxp13          maximum number of atoms in groups 1-3 to an atom
maxp14          maximum number of atoms in groups 1-4 to an atom
np11            number of atoms in polarization group of each atom
np12            number of atoms in groups 1-2 to each atom
np13            number of atoms in groups 1-3 to each atom
np14            number of atoms in groups 1-4 to each atom
ip11            atom numbers of atoms in same group as each atom
ip12            atom numbers of atoms in groups 1-2 to each atom
ip13            atom numbers of atoms in groups 1-3 to each atom
ip14            atom numbers of atoms in groups 1-4 to each atom
```

POLOPT Module induced dipoles for OPT extrapolation

```
maxopt          maximum order for OPT induced dipole extrapolation
optorder        highest coefficient order for OPT dipole extrapolation
```

(continues on next page)

(continued from previous page)

optlevel	current OPT order for reciprocal potential and field
copt	coefficients for OPT total induced dipole moments
copm	coefficients for OPT incremental induced dipole moments
uopt	OPT induced dipole components at each multipole site
uoptp	OPT induced dipoles in field used for energy terms
uopts	OPT GK or PB induced dipoles at each multipole site
uoptps	OPT induced dipoles in field used for GK or PB energy
fopt	OPT fractional reciprocal potentials at multipole sites
foptp	OPT fractional reciprocal potentials for energy terms

POLPCG Module induced dipoles via the PCG solver

mindex	index into preconditioner inverse for PCG solver
pcgpeek	value of acceleration factor for PCG peek step
minv	preconditioner inverse for induced dipole PCG solver
pcgprec	flag to allow use of preconditioner with PCG solver
pcgguess	flag to use initial PCG based on direct field

POLPOT Module polarization functional form details

politer	maximum number of induced dipole SCF iterations
poleps	induced dipole convergence criterion (rms Debye/atom)
p2scale	scale factor for 1-2 polarization energy interactions
p3scale	scale factor for 1-3 polarization energy interactions
p4scale	scale factor for 1-4 polarization energy interactions
p5scale	scale factor for 1-5 polarization energy interactions
p2iscale	scale factor for 1-2 intragroup polarization energy
p3iscale	scale factor for 1-3 intragroup polarization energy
p4iscale	scale factor for 1-4 intragroup polarization energy
p5iscale	scale factor for 1-5 intragroup polarization energy
d1scale	scale factor for intra-group direct induction
d2scale	scale factor for 1-2 group direct induction
d3scale	scale factor for 1-3 group direct induction
d4scale	scale factor for 1-4 group direct induction
u1scale	scale factor for intra-group mutual induction
u2scale	scale factor for 1-2 group mutual induction
u3scale	scale factor for 1-3 group mutual induction
u4scale	scale factor for 1-4 group mutual induction
w2scale	scale factor for 1-2 induced dipole interactions
w3scale	scale factor for 1-3 induced dipole interactions
w4scale	scale factor for 1-4 induced dipole interactions
w5scale	scale factor for 1-5 induced dipole interactions
udiag	acceleration factor for induced dipole SCF iterations
dpequal	flag to set dscale values equal to pscale values
use_thole	flag to use Thole damped polarization interactions
use_dirdamp	flag to use damped direct polarization interactions
poltyp	type of polarization (MUTUAL, DIRECT, OPT or TCG)

POLTCG Module induced dipoles via the TCG solver

tcgorder	total number of TCG iterations to be used

(continues on next page)

(continued from previous page)

```
tcgnab          number of mutual induced dipole components
tcgpeek         value of acceleration factor for TCG peek step
uad             left-hand side mutual induced d-dipoles
uap             left-hand side mutual induced p-dipoles
ubd             right-hand side mutual induced d-dipoles
ubp             right-hand side mutual induced p-dipoles
tcgguess        flag to use initial TCG based on direct field
```

POTENT Module usage of potential energy components

```
use_bond        logical flag governing use of bond stretch potential
use_angle       logical flag governing use of angle bend potential
use_strbnd      logical flag governing use of stretch-bend potential
use_urey        logical flag governing use of Urey-Bradley potential
use_angang      logical flag governing use of angle-angle cross term
use_opbend      logical flag governing use of out-of-plane bend term
use_opdist      logical flag governing use of out-of-plane distance
use_improp      logical flag governing use of improper dihedral term
use_imptor      logical flag governing use of improper torsion term
use_tors        logical flag governing use of torsional potential
use_pitors      logical flag governing use of pi-system torsion term
use_strtor      logical flag governing use of stretch-torsion term
use_angtor      logical flag governing use of angle-torsion term
use_tortor      logical flag governing use of torsion-torsion term
use_vdw         logical flag governing use of vdw der Waals potential
use_repuls      logical flag governing use of Pauli repulsion term
use_disp        logical flag governing use of dispersion potential
use_charge      logical flag governing use of charge-charge potential
use_chgdpl      logical flag governing use of charge-dipole potential
use_dipole      logical flag governing use of dipole-dipole potential
use_mpole       logical flag governing use of multipole potential
use_polar       logical flag governing use of polarization term
use_chgtrn      logical flag governing use of charge transfer term
use_rxnfld      logical flag governing use of reaction field term
use_solv        logical flag governing use of continuum solvation term
use_metal       logical flag governing use of ligand field term
use_geom        logical flag governing use of geometric restraints
use_extra       logical flag governing use of extra potential term
use_born        logical flag governing use of Born radii values
use_orbit       logical flag governing use of pisystem computation
```

POTFIT Module values for electrostatic potential fit

```
nconf           total number of configurations to be analyzed
namax           maximum number of atoms in the largest configuration
ngatm           total number of atoms with active potential grid points
nfatm           total number of atoms in electrostatic potential fit
npgrid          total number of electrostatic potential grid points
ipgrid          atom associated with each potential grid point
resp            weight used to restrain parameters to original values
xdpl0           target x-component of the molecular dipole moment
```

(continues on next page)

(continued from previous page)

ydpl0	target y-component of the molecular dipole moment
zdpl0	target z-component of the molecular dipole moment
xxqdp0	target xx-component of the molecular quadrupole moment
xyqdp0	target xy-component of the molecular quadrupole moment
xzqdp0	target xz-component of the molecular quadrupole moment
yyqdp0	target yy-component of the molecular quadrupole moment
yzqdp0	target yz-component of the molecular quadrupole moment
zzqdp0	target zz-component of the molecular quadrupole moment
fit0	initial value of each parameter used in potential fit
fchg	partial charges by atom type during potential fit
fpol	atomic multipoles by atom type during potential fit
pgrid	Cartesian coordinates of potential grid points
epot	values of electrostatic potential at grid points
use_dpl	flag to include molecular dipole in potential fit
use_qdp	flag to include molecular quadrupole in potential fit
fit_mpl	flag for atomic monopoles to vary in potential fit
fit_dpl	flag for atomic dipoles to vary in potential fit
fit_qdp	flag for atomic quadrupoles to vary in potential fit
fitchg	flag marking atom types for use in partial charge fit
fitpol	flag marking atom types for use in atomic multipole fit
gatm	flag to use potential grid points around each atom
fatm	flag to use each atom in electrostatic potential fit

PTABLE Module symbols and info for chemical elements

maxele	maximum number of elements from periodic table
atmass	standard atomic weight for each chemical element
vdwrad	van der Waals radius for each chemical element
covrad	covalent radius for each chemical element
elemnt	atomic symbol for each chemical element

REFER Module reference atomic coordinate storage

nref	total number of atoms in each reference system
refltitle	length in characters of each reference title line
refleng	length in characters of each reference filename
reftyp	atom types of the atoms in each reference system
n12ref	number of atoms bonded to each reference atom
i12ref	atom numbers of atoms 1-2 connected to each atom
xboxref	reference a-axis length of periodic box
yboxref	reference b-axis length of periodic box
zboxref	reference c-axis length of periodic box
alpharef	reference angle between b- and c-axes of box
betaref	reference angle between a- and c-axes of box
gammaref	reference angle between a- and b-axes of box
xref	reference x-coordinates for atoms in each system
yref	reference y-coordinates for atoms in each system
zref	reference z-coordinates for atoms in each system
refnam	atom names of the atoms in each reference system
reffile	base filename for each reference system
reftitle	title used to describe each reference system

REPEL Module Pauli repulsion for current structure

```
nrep            total number of repulsion sites in the system
sizpr           Pauli repulsion size parameter value at each site
dmppr           Pauli repulsion alpha damping value at each site
elepr           Pauli repulsion valence electrons at each site
```

REPPOT Module repulsion interaction scale factors

```
r2scale         scale factor for 1-2 repulsion energy interactions
r3scale         scale factor for 1-3 repulsion energy interactions
r4scale         scale factor for 1-4 repulsion energy interactions
r5scale         scale factor for 1-5 repulsion energy interactions
```

RESDUE Module amino acid & nucleotide residue names

```
maxamino        maximum number of amino acid residue types
maxnuc          maximum number of nucleic acid residue types
ntyp            biotypes for mid-chain peptide backbone N atoms
catyp           biotypes for mid-chain peptide backbone CA atoms
ctyp            biotypes for mid-chain peptide backbone C atoms
hntyp           biotypes for mid-chain peptide backbone HN atoms
otyp            biotypes for mid-chain peptide backbone O atoms
hatyp           biotypes for mid-chain peptide backbone HA atoms
cbtyp           biotypes for mid-chain peptide backbone CB atoms
nntyp           biotypes for N-terminal peptide backbone N atoms
cantyp          biotypes for N-terminal peptide backbone CA atoms
cntyp           biotypes for N-terminal peptide backbone C atoms
hnntyp          biotypes for N-terminal peptide backbone HN atoms
ontyp           biotypes for N-terminal peptide backbone O atoms
hantyp          biotypes for N-terminal peptide backbone HA atoms
nctyp           biotypes for C-terminal peptide backbone N atoms
cactyp          biotypes for C-terminal peptide backbone CA atoms
cctyp           biotypes for C-terminal peptide backbone C atoms
hnctyp          biotypes for C-terminal peptide backbone HN atoms
octyp           biotypes for C-terminal peptide backbone O atoms
hactyp          biotypes for C-terminal peptide backbone HA atoms
o5typ           biotypes for nucleotide backbone and sugar O5' atoms
c5typ           biotypes for nucleotide backbone and sugar C5' atoms
h51typ          biotypes for nucleotide backbone and sugar H5' atoms
h52typ          biotypes for nucleotide backbone and sugar H5'' atoms
c4typ           biotypes for nucleotide backbone and sugar C4' atoms
h4typ           biotypes for nucleotide backbone and sugar H4' atoms
o4typ           biotypes for nucleotide backbone and sugar O4' atoms
c1typ           biotypes for nucleotide backbone and sugar C1' atoms
h1typ           biotypes for nucleotide backbone and sugar H1' atoms
c3typ           biotypes for nucleotide backbone and sugar C3' atoms
h3typ           biotypes for nucleotide backbone and sugar H3' atoms
c2typ           biotypes for nucleotide backbone and sugar C2' atoms
h21typ          biotypes for nucleotide backbone and sugar H2' atoms
o2typ           biotypes for nucleotide backbone and sugar O2' atoms
h22typ          biotypes for nucleotide backbone and sugar H2'' atoms
o3typ           biotypes for nucleotide backbone and sugar O3' atoms
```

(continues on next page)

(continued from previous page)

```
ptyp            biotypes for nucleotide backbone and sugar P atoms
optyp           biotypes for nucleotide backbone and sugar OP atoms
h5ttyp          biotypes for nucleotide backbone and sugar H5T atoms
h3ttyp          biotypes for nucleotide backbone and sugar H3T atoms
amino           three-letter abbreviations for amino acids types
nuclz           three-letter abbreviations for nucleic acids types
amino1          one-letter abbreviations for amino acids types
nuclz1          one-letter abbreviations for nucleic acids types
```

RESTRN Module parameters for geometrical restraints

```
npfix           number of position restraints to be applied
ndfix           number of distance restraints to be applied
nafix           number of angle restraints to be applied
ntfix           number of torsional restraints to be applied
ngfix           number of group distance restraints to be applied
nchir           number of chirality restraints to be applied
ipfix           atom number involved in each position restraint
kpfix           flags to use x-, y-, z-coordinate position restraints
idfix           atom numbers defining each distance restraint
iafix           atom numbers defining each angle restraint
itfix           atom numbers defining each torsional restraint
igfix           group numbers defining each group distance restraint
ichir           atom numbers defining each chirality restraint
depth           depth of shallow Gaussian basin restraint
width           exponential width coefficient of Gaussian basin
rwall           radius of spherical droplet boundary restraint
xpfix           x-coordinate target for each restrained position
ypfix           y-coordinate target for each restrained position
zpfix           z-coordinate target for each restrained position
pfix            force constant and flat-well range for each position
dfix            force constant and target range for each distance
afix            force constant and target range for each angle
tfix            force constant and target range for each torsion
gfix            force constant and target range for each group distance
chir            force constant and target range for chiral centers
use_basin       logical flag governing use of Gaussian basin
use_wall        logical flag governing use of droplet boundary
```

RGDDYN Module rigid body MD velocities and momenta

```
xcmo            x-component from each atom to center of rigid body
ycmo            y-component from each atom to center of rigid body
zcmo            z-component from each atom to center of rigid body
vcm             current translational velocity of each rigid body
wcm             current angular velocity of each rigid body
lm              current angular momentum of each rigid body
vc              half-step translational velocity for kinetic energy
wc              half-step angular velocity for kinetic energy
linear          logical flag to mark group as linear or nonlinear
```

RIGID Module rigid body coordinates for atom groups

```
xrb             rigid body reference x-coordinate for each atom
yrb             rigid body reference y-coordinate for each atom
zrb             rigid body reference z-coordinate for each atom
rbc             current rigid body coordinates for each group
use_rigid       flag to mark use of rigid body coordinate system
```

RING Module **number and location of ring structures**

```
nring3          total number of 3-membered rings in the system
nring4          total number of 4-membered rings in the system
nring5          total number of 5-membered rings in the system
nring6          total number of 6-membered rings in the system
nring7          total number of 7-membered rings in the system
iring3          numbers of the atoms involved in each 3-ring
iring4          numbers of the atoms involved in each 4-ring
iring5          numbers of the atoms involved in each 5-ring
iring6          numbers of the atoms involved in each 6-ring
iring7          numbers of the atoms involved in each 7-ring
```

ROTBND Module **molecule partitions for bond rotation**

```
nrot            total number of atoms moving when bond rotates
rot             atom numbers of atoms moving when bond rotates
use_short       logical flag governing use of shortest atom list
```

RXNFLD Module **reaction field matrix and indices**

```
ijk             indices into the reaction field element arrays
b1              first reaction field matrix element array
b2              second reaction field matrix element array
```

RXNPOT Module **reaction field functional form details**

```
rfsize          radius of reaction field sphere centered at origin
rfbulkd         bulk dielectric constant of reaction field continuum
rfterms         number of terms to use in reaction field summation
```

SCALES Module **optimization parameter scale factors**

```
scale           multiplicative factor for each optimization parameter
set_scale       logical flag to show if scale factors have been set
```

SEQUEN Module **sequence information for biopolymer**

```
nseq            total number of residues in biopolymer sequences
nchain          number of separate biopolymer sequence chains
ichain          first and last residue in each biopolymer chain
seqtyp          residue type for each residue in the sequence
seq             three-letter code for each residue in the sequence
chnnam          one-letter identifier for each sequence chain
chntyp          contents of each chain (GENERIC, PEPTIDE or NUCLEIC)
```

SHUNT Module **polynomial switching function values**

```
off             distance at which the potential energy goes to zero
off2            square of distance at which the potential goes to zero
cut             distance at which switching of the potential begins
cut2            square of distance at which the switching begins
c0              zeroth order coefficient of multiplicative switch
c1              first order coefficient of multiplicative switch
c2              second order coefficient of multiplicative switch
c3              third order coefficient of multiplicative switch
c4              fourth order coefficient of multiplicative switch
c5              fifth order coefficient of multiplicative switch
f0              zeroth order coefficient of additive switch function
f1              first order coefficient of additive switch function
f2              second order coefficient of additive switch function
f3              third order coefficient of additive switch function
f4              fourth order coefficient of additive switch function
f5              fifth order coefficient of additive switch function
f6              sixth order coefficient of additive switch function
f7              seventh order coefficient of additive switch function
```

SIZES Module **parameters to set array dimensions**

```
sizes"          sets values for critical array dimensions used
throughout      the software; these parameters fix the size of
the             largest systems that can be handled
parameter       maximum allowed number of:
maxatm          atoms in the molecular system
maxtyp          force field atom type definitions
maxclass        force field atom class definitions
maxval          atoms directly bonded to an atom
maxref          stored reference molecular systems
maxgrp          user-defined groups of atoms
maxres          residues in the macromolecule
maxfix          geometric constraints and restraints
```

SOCKET Module **socket communication control parameters**

```
skttyp          socket information type (1=DYN, 2=OPT)
cstep           current dynamics or optimization step number
cdt             current dynamics cumulative simulation time
cenergy         current potential energy from simulation
sktstart        logical flag to indicate socket initialization
sktstop         logical flag to indicate socket shutdown
use_socket      logical flag governing use of external sockets
```

SOLUTE Module **continuum solvation model parameters**

```
doffset         dielectric offset to continuum solvation atomic radii
p1              single-atom scale factor for analytical Still radii
p2              1-2 interaction scale factor for analytical Still radii
p3              1-3 interaction scale factor for analytical Still radii
p4              nonbonded scale factor for analytical Still radii
p5              soft cutoff parameter for analytical Still radii
```

(continues on next page)

(continued from previous page)

```
rsolv           atomic radius of each atom for continuum solvation
asolv           atomic surface area solvation parameters
rborn           Born radius of each atom for GB/SA solvation
drb             solvation derivatives with respect to Born radii
drbp            GK polarization derivatives with respect to Born radii
drobc           chain rule term for Onufriev-Bashford-Case radii
gpol            polarization self-energy values for each atom
shct            overlap scale factors for Hawkins-Cramer-Truhlar radii
aobc            alpha values for Onufriev-Bashford-Case radii
bobc            beta values for Onufriev-Bashford-Case radii
gobc            gamma values for Onufriev-Bashford-Case radii
vsolv           atomic volume of each atom for use with ACE
wace            "omega" values for atom class pairs for use with ACE
s2ace           "sigma^2" values for atom class pairs for use with ACE
uace            "mu" values for atom class pairs for use with ACE
solvtyp         type of continuum solvation energy model in use
borntyp         method to be used for the Born radius computation
```

STODYN Module SD trajectory frictional coefficients

```
friction        global frictional coefficient for exposed particle
fgamma          atomic frictional coefficients for each atom
use_sdarea      logical flag to use surface area friction scaling
```

STRBND Module stretch-bends in current structure

```
nstrbnd         total number of stretch-bend interactions
isb             angle and bond numbers used in stretch-bend
sbk             force constants for stretch-bend terms
```

STRTOR Module stretch-torsions in current structure

```
nstrtor         total number of stretch-torsion interactions
ist             torsion and bond numbers used in stretch-torsion
kst             1-, 2- and 3-fold stretch-torsion force constants
```

SYNTRN Module synchronous transit path definition

```
tpath           value of the path coordinate (0=reactant, 1=product)
ppath           path coordinate for extra point in quadratic transit
xmin1           reactant coordinates as array of optimization variables
xmin2           product coordinates as array of optimization variables
xm              extra coordinate set for quadratic synchronous transit
```

TARRAY Module store dipole-dipole matrix elements

```
ntpair          number of stored dipole-dipole matrix elements
tindex          index into stored dipole-dipole matrix values
tdipdip         stored dipole-dipole matrix element values
```

TITLES Module title for current molecular system

ltitle	length in characters of the nonblank title string
title	title used to describe the current structure

TORPOT Module torsional functional form details

idihunit	convert improper dihedral energy to kcal/mole
itorunit	convert improper torsion amplitudes to kcal/mole
torsunit	convert torsional parameter amplitudes to kcal/mole
ptorunit	convert pi-system torsion energy to kcal/mole
storunit	convert stretch-torsion energy to kcal/mole
atorunit	convert angle-torsion energy to kcal/mole
ttorunit	convert torsion-torsion energy to kcal/mole

TORS Module torsional angles in current structure

ntors	total number of torsional angles in the system
itors	numbers of the atoms in each torsional angle
tors1	1-fold amplitude and phase for each torsional angle
tors2	2-fold amplitude and phase for each torsional angle
tors3	3-fold amplitude and phase for each torsional angle
tors4	4-fold amplitude and phase for each torsional angle
tors5	5-fold amplitude and phase for each torsional angle
tors6	6-fold amplitude and phase for each torsional angle

TORTOR Module torsion-torsions in current structure

ntortor	total number of torsion-torsion interactions
itt	atoms and parameter indices for torsion-torsion

TREE Module potential smoothing search tree levels

maxpss	maximum number of potential smoothing levels
nlevel	number of levels of potential smoothing used
etree	energy reference value at the top of the tree
ilevel	smoothing deformation value at each tree level

UNITS Module physical constants and unit conversions

literature	references:
D.	B. Newell, F. Cabiati, J. Fischer, K. Fujii, S. G. Karshenboim,
S.	Margolis, E. de Mirandes, P. J. Mohr, F. Nez, K. Pachucki,
T.	J. Quinn, N. Taylor, M. Wang, B. M. Wood and Z. Zhang, "The
CODATA	2017 Values of h, e, k, and Na for the Revision of the SI",
Metrologia	55, L13-L16 (2018)
P.	J. Mohr, D. B. Newell and B. N. Taylor, "CODATA Recommended
Values	of the Fundamental Physical Constants: 2014", Journal of
Physical	and Chemical Reference Data, 45, 043102 (2016)
Where	available, values are from the 2017 CODATA adjustment
based	on exact physical constants for the revised SI
Other	values are from the 2014 CODATA reference constants; also
available	online from the National Institute of Standards and
Technology	at http://physics.nist.gov/cuu/Constants/index.html/

(continues on next page)

(continued from previous page)

```
The             conversion from calorie to Joule is the definition of the
thermochemical  calorie as 1 cal = 4.1840 J from ISO 31-4 (1992)
The             "coulomb" energy conversion factor is found by dimensional
analysis        of Coulomb's Law, ie, by dividing the square of the
elementary      charge in Coulombs by 4*pi*eps0*rij, where eps0 is
the             permittivity of vacuum (the "electric constant"); note that
eps0            is typically given in F/m, equivalent to C**2/(J-m)
The             approximate value used for the Debye, 3.33564 x 10-30 C-m,
is              from IUPAC Compendium of Chemical Technology, 2nd Ed. (1997)
The             value of "prescon" is based on definition of 1 atmosphere
as              101325 Pa set by the 10th Conference Generale des Poids et
Mesures         (1954), where a Pascal (Pa) is equal to a J/m**3
avogadro        Avogadro's number (N) in particles/mole
lightspd        speed of light in vacuum (c) in cm/ps
boltzmann       Boltzmann constant (kB) in g*Ang**2/ps**2/mole/K
gasconst        ideal gas constant (R) in kcal/mole/K
elemchg         elementary charge of a proton in Coulombs
vacperm         vacuum permittivity (electric constant, eps0) in F/m
emass           mass of an electron in atomic mass units
planck          Planck's constant (h) in J-s
joule           conversion from calorie to joule
ekcal           conversion from kcal to g*Ang**2/ps**2
bohr            conversion from Bohr to Angstrom
hartree         conversion from Hartree to kcal/mole
evolt           conversion from Hartree to electron-volt
efreq           conversion from Hartree to cm-1
coulomb         conversion from electron**2/Ang to kcal/mole
debye           conversion from electron-Ang to Debye
prescon         conversion from kcal/mole/Ang**3 to Atm
```

UPRIOR Module previous values of induced dipoles

```
maxpred         maximum number of predictor induced dipoles to save
nualt           number of sets of prior induced dipoles in storage
maxualt         number of sets of induced dipoles needed for predictor
gear            coefficients for Gear predictor binomial method
aspc            coefficients for always stable predictor-corrector
bpred           coefficients for induced dipole predictor polynomial
bpredp          coefficients for predictor polynomial in energy field
bpreds          coefficients for predictor for PB/GK solvation
bpredps         coefficients for predictor in PB/GK energy field
udalt           prior values for induced dipoles at each site
upalt           prior values for induced dipoles in energy field
usalt           prior values for induced dipoles for PB/GK solvation
upsalt          prior values for induced dipoles in PB/GK energy field
use_pred        flag to control use of induced dipole prediction
polpred         type of predictor polynomial (GEAR, ASPC or LSQR)
```

UREY Module Urey-Bradley interactions in structure

```
nurey           total number of Urey-Bradley terms in the system
```

(continues on next page)

(continued from previous page)

iury	numbers of the atoms in each Urey-Bradley interaction
uk	Urey-Bradley force constants (kcal/mole/Ang**2)
ul	ideal 1-3 distance values in Angstroms

URYPOT Module Urey-Bradley functional form details

cury	cubic coefficient in Urey-Bradley potential
qury	quartic coefficient in Urey-Bradley potential
ureyunit	convert Urey-Bradley energy to kcal/mole

USAGE Module atoms active during energy computation

nuse	total number of active atoms in energy calculation
iuse	numbers of the atoms active in energy calculation
use	true if an atom is active, false if inactive

VALFIT Module valence term parameter fitting values

fit_bond	logical flag to fit bond stretch parameters
fit_angle	logical flag to fit angle bend parameters
fit_strbnd	logical flag to fit stretch-bend parameters
fit_urey	logical flag to fit Urey-Bradley parameters
fit_opbend	logical flag to fit out-of-plane bend parameters
fit_tors	logical flag to fit torsional parameters
fit_struct	logical flag to structure-fit valence parameters
fit_force	logical flag to force-fit valence parameters

VDW Module van der Waals terms in current structure

nvdw	total number van der Waals active sites in the system
ivdw	number of the atom for each van der Waals active site
jvdw	type or class index into vdw parameters for each atom
ired	attached atom from which reduction factor is applied
kred	value of reduction factor parameter for each atom
xred	reduced x-coordinate for each atom in the system
yred	reduced y-coordinate for each atom in the system
zred	reduced z-coordinate for each atom in the system
radmin	minimum energy distance for each atom class pair
epsilon	well depth parameter for each atom class pair
radmin4	minimum energy distance for 1-4 interaction pairs
epsilon4	well depth parameter for 1-4 interaction pairs
radhbnd	minimum energy distance for hydrogen bonding pairs
epshbnd	well depth parameter for hydrogen bonding pairs

VDWPOT Module van der Waals functional form details

igauss	coefficients of Gaussian fit to vdw potential
ngauss	number of Gaussians used in fit to vdw potential
abuck	value of "A" constant in Buckingham vdw potential
bbuck	value of "B" constant in Buckingham vdw potential
cbuck	value of "C" constant in Buckingham vdw potential

(continues on next page)

(continued from previous page)

ghal	value of "gamma" in buffered 14-7 vdw potential
dhal	value of "delta" in buffered 14-7 vdw potential
v2scale	factor by which 1-2 vdw interactions are scaled
v3scale	factor by which 1-3 vdw interactions are scaled
v4scale	factor by which 1-4 vdw interactions are scaled
v5scale	factor by which 1-5 vdw interactions are scaled
use_vcorr	flag to use long range van der Waals correction
vdwindex	indexing mode (atom type or class) for vdw parameters
vdwtyp	type of van der Waals potential energy function
radtyp	type of parameter (sigma or R-min) for atomic size
radsiz	atomic size provided as radius or diameter
radrule	combining rule for atomic size parameters
epsrule	combining rule for vdw well depth parameters
gausstyp	type of Gaussian fit to van der Waals potential

VIBS Module iterative vibrational analysis components

rho	trial vectors for iterative vibrational analysis
rhok	alternate vectors for iterative vibrational analysis
rwork	temporary work array for eigenvector transformation

VIRIAL Module components of internal virial tensor

vir	total internal virial Cartesian tensor components
use_virial	logical flag governing use of virial computation

WARP Module potential surface smoothing parameters

deform	value of the smoothing deformation parameter
difft	diffusion coefficient for torsional potential
diffv	diffusion coefficient for van der Waals potential
diffc	diffusion coefficient for charge-charge potential
m2	second moment of the GDA gaussian for each atom
use_smooth	flag to use a potential energy smoothing method
use_dem	flag to use diffusion equation method potential
use_gda	flag to use gaussian density annealing potential
use_tophat	flag to use analytical tophat smoothed potential
use_stophat	flag to use shifted tophat smoothed potential

XTALS Module structures used for parameter fitting

maxlsq	maximum number of least squares variables
maxrsd	maximum number of residual functions
nxtal	number of molecular structures to be stored
nvary	number of potential parameters to optimize
ivary	index for the types of potential parameters
iresid	structure to which each residual function refers
vary	atom numbers involved in potential parameters
e0_lattice	ideal lattice energy for the current crystal
vartyp	type of each potential parameter to be optimized
rsdtyp	experimental variable for each of the residuals

ZCLOSE Module Z-matrix ring openings and closures

```
nadd            number of added bonds between Z-matrix atoms
ndel            number of bonds between Z-matrix bonds to delete
iadd            numbers of the atom pairs defining added bonds
idel            numbers of the atom pairs defining deleted bonds
```

ZCOORD Module Z-matrix internal coordinate values

```
iz              defining atom numbers for each Z-matrix atom
zbond           bond length used to define each Z-matrix atom
zang            bond angle used to define each Z-matrix atom
ztors           angle or torsion used to define Z-matrix atom
```

CHAPTER

ELEVEN

TEST CASES & EXAMPLES

This section contains brief descriptions of the sample calculations found in the EXAMPLE subdirectory of the Tinker distribution. These examples exercise several of the current Tinker programs and are intended to provide a flavor of the capabilities of the package.

ANION Test

Computes an estimation of the free energy of hydration of Cl- anion vs. Br- anion via a 2 picosecond simulation on a "hybrid" anion in a box of water followed by a free energy perturbation calculation.

ARGON Test

Performs an initial energy minimization on a periodic box containing 150 argon atoms followed by 6 picoseconds of a molecular dynamics using a modified Beeman integration algorithm and a Bersedsen thermostat.

CLUSTER Test

Performs a set of 10 Gaussian density annealing (GDA) trials on a cluster of 13 argon atoms in an attempt to locate the global minimum energy structure.

CRAMBIN Test

Generates a Tinker file from a PDB file, followed by a single point energy computation and determination of the molecular volume and surface area.

CYCLOHEX Test

First approximately locates the transition state between chair and boat cyclohexane, followed by subsequent refinement of the transition state and a final vibrational analysis to show that a single negative frequency is associated with the saddle point.

DHFR Test

Performs 10 steps of molecular dynamics on a pre-equilibrated system of DHFR protein in a box or water using the AMOEBA force field. Note this test case is the so-called Joint Amber-CHARMM "JAC" benchmark containing 23558 total atoms.

DIALANINE Test

Finds all the local minima of alanine dipeptide via a potential energy surface scan using torsional modes to jump between the minima.

ENKEPHALIN Test

Produces coordinates from the met-enkephalin amino acid sequence and phi/psi angles, followed by truncated Newton energy minimization and determination of the lowest frequency normal mode.

ETHANOL Test

Performs fitting of torsional parameter values for the ethanol C-C-O-H bond based on relative quantum mechanical (G09) energies for rotating the C-O bond.

FORMAMIDE Test

Generates a unit cell from fractional coordinates, followed by full crystal energy minimization and determination of optimal carbonyl oxygen energy parameters from a fit to lattice energy and structure.

GPCR Test

Finds the lowest-frequency normal mode of bacteriorhodopsin using vibrational analysis via a sliding block iterative matrix diagonalization. Alter the gpcr.run script to save the file gpcr.001 for later viewing of the mode.

HELIX Test

Performs a rigid-body optimization of the packing of two idealized polyalanine helices using only van der Waals interactions.

ICE Test

Performs a short MD simulation of the monoclinic ice V crystal form using the iAMOEBA water model, pairwise neighbor lists and PME electrostatics.

IFABP Test

Generates three distance geometry structures for intestinal fatty acid binding protein from a set of NOE distance restraints and torsional restraints.

METHANOL Test

Processes distributed multipole analysis (DMA) output to extract coordinates and permanent multipoles, set local frames and polarization groups, remove intramolecular polarization, detect and average equivalent atomic sites.

NITROGEN Test

Calculates the self-diffusion constant and the N-N radial distribution function for liquid nitrogen via analysis of a 50ps MD trajectory.

SALT Test

Converts a sodium chloride assymetric unit to the corresponding unit cell, then runs a crystal minimization starting from the initial diffraction structure using Ewald summation to model the long-range electrostatic interactions.

TETRAALA Test

Generates capped alanine tetrapeptide in an extended conformation, then use Monte Carlo Minimization with random torsional moves to find the global minimum energy structure.

WATER Test

Fits the electrostatic potential around an AMOEBA water molecule to the QM-derived potential (MP2/aug-cc-pVTZ) on a grid of points outside the molecular surface.

CHAPTER

TWELVE

BENCHMARK RESULTS

The tables in this section provide CPU benchmarks for basic Tinker energy and derivative evaluations, vibrational analysis and molecular dynamics simulation. All times are in seconds and were measured with Tinker executables dimensioned to a maximum of 1000000 atoms. Each benchmark was run on an unloaded machine and is the fastest time reported for that particular machine. The first five benchmarks are run serial on a single thread, while the last four benchmarks reflect OpenMP parallel performance. If you have built Tinker on an alternative machine type and are able to run the benchmarks on the additional machine type, please send the results for inclusion in a future listing.

12.1 Calmodulin Energy Evaluation (Serial)

Gas-Phase Calmodulin Molecule, 2264 Atoms, Amber ff94 Force Field, No Nonbonded Cutoffs, 100 Evaluations

Machine Type (OS/Compiler)	CPU	Energy	Gradient	Hessian
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	26.2	50.9	149.9
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	17.6	34.7	106.8
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12.2	32.9	95.4
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	7.2	16.6	50.4
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	7.3	16.7	51.5
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	5.4	10.4	33.2

12.2 Crambin Crystal Energy Evaluation (Serial)

Crambin Unit Cell, 1360 Atoms in Periodic Unit Cell, OPLS-UA Force Field with PME Electrostatics, 9.0 Ang vdw Cutoff, 1000 Evaluations

Machine Type (OS/Compiler)	CPU	Energy	Gradient	Hessian
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	36.6	54.1	220.5
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	34.6	47.8	190.8
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	31.3	43.8	161.0
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	19.1	26.0	85.0

(continues on next page)

(continued from previous page)

MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	18.8	26.6	87.8
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	15.6	19.9	67.9

12.3 Crambin Normal Mode Calculation (Serial)

Hessian Eigenvalues, Normal Modes and Vibrational Freqencies for the 42-Amino Acid, 642-Atom Protein Crambin, CHARMM-22 Force Field with Cutoffs

Machine Type (OS/Compiler)	CPU	Seconds
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	41.1
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	40.5
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	23.5
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	15.9
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	15.3
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	11.0

12.4 Water Box Molecular Dynamics using TIP3P (Serial)

MD run of 10000 Steps for 216 TIP3P Waters in 18.643 Ang Periodic Box, 9.0 Ang Shifted & Switched Cutoffs, Rattle for Rigid TIP3P, 1.0 fs Time Step with Modified Beeman Integrator

Machine Type (OS/Compiler)	CPU	Seconds
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	214.0
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	152.1
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	158.1
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	80.4
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	82.1
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	55.1

12.5 Water Box Molecular Dynamics using AMOEBA (Serial)

MD run of 1000 Steps for 216 AMOEBA Waters in a 18.643 Ang Box, Neighbor Lists, PME with a 20x20x20 FFT and 7.0 Ang Real-Space Cutoff, 9.0 Ang vdW Cutoff with Correction, 1.0 fs Time Step with Modified Beeman Integrator, and 0.00001 RMS Induced Dipole Convergence

Machine Type (OS/Compiler)	CPU	Seconds
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	112.8
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	99.7
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	82.7
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	46.9
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	48.7
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	37.0

12.6 MD on DHFR in Water using CHARMM (OpenMP Parallel)

MD run of 100 Steps for CHARMM DHFR in Water (23558 Atoms, 62.23 Ang Box), Neighbor Lists, PME with a 64x64x64 FFT and 7.0 Ang Real-Space Cutoff, 9.0 Ang vdW Cutoff, 1.0 fs Time Step with Modified Beeman Integrator; OpenMP timings as "wall clock" time, with parallel speedup in parentheses

Machine Type (OS/Compiler)	CPU	Core/Thread	Seconds
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/1	105.8 (1.00)
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/4	68.1 (1.55)
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/1	88.3 (1.00)
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/8	49.1 (1.80)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/1	76.6 (1.00)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/24	33.9 (2.26)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/1	44.7 (1.00)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/2	32.5 (1.38)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/4	25.0 (1.79)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/1	45.4 (1.00)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/2	34.3 (1.32)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/4	26.9 (1.69)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/8	23.8 (1.91)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/1	34.5 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/2	23.6 (1.46)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/4	16.8 (2.05)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/6	14.5 (2.38)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/12	14.1 (2.45)

12.7 MD on DHFR in Water using AMOEBA (OpenMP Parallel)

MD run of 100 Steps for AMOEBA DHFR in Water (23558 Atoms, 62.23 Ang Box), Neighbor Lists, PME with a 64x64x64 FFT and 7.0 Ang Real-Space Cutoff, 9.0 Ang vdW Cutoff with Correction, 1.0 fs Time Step with Modified Beeman Integrator, and 0.00001 RMS Induced Dipole Convergence; OpenMP timings reported as "wall clock" time, with parallel speedup in parentheses

Machine Type (OS/Compiler)	CPU	Core/Thread	Seconds
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/1	507.5 (1.00)
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/4	246.3 (2.06)
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/1	432.2 (1.00)
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/8	158.0 (2.74)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/1	357.6 (1.00)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/24	98.3 (3.64)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/1	202.6 (1.00)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/2	143.5 (1.41)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/4	92.2 (2.20)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/1	219.1 (1.00)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/2	159.8 (1.37)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/4	98.6 (2.22)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/8	85.0 (2.58)

(continues on next page)

(continued from previous page)

Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/1	153.3 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/2	108.0 (1.42)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/4	69.7 (2.20)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/6	57.1 (2.68)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/12	59.5 (2.58)

12.8 MD on COX-2 in Water using OPLS-AA (OpenMP Parallel)

MD run of 100 Steps for OPLS-AA COX-2 in Water (174219 Atoms, 120.0 Ang Box), Neighbor Lists, PME with a 128x128x128 FFT and 7.0 Ang Real-Space Cutoff, 9.0 Ang vdW Cutoff, 1.0 fs Time Step with Modified Beeman Integrator; RATTLE for all X-H bonds and rigid TIP3P Water; OpenMP timings reported as "wall clock" time, with parallel speedup in parentheses

Machine Type (OS/Compiler)	CPU	Core/Thread	Seconds
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/1	798.6 (1.00)
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/4	487.2 (1.65)
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/1	698.5 (1.00)
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/8	392.8 (1.78)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/1	543.6 (1.00)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/24	240.2 (2.26)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/1	344.2 (1.00)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/2	270.9 (1.27)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/4	196.8 (1.75)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/1	347.6 (1.00)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/2	276.1 (1.26)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/4	193.7 (1.79)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/8	173.2 (2.01)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/1	262.3 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/2	202.4 (1.30)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/4	143.2 (1.83)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/6	127.2 (2.06)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/12	120.9 (2.17)

12.9 MD on COX-2 in Water using AMOEBA (OpenMP Parallel)

MD run of 100 Steps for AMOEBA COX-2 in Water (174219 Atoms, 120.0 Ang Box), Neighbor Lists, PME with a 128x128x128 FFT and 7.0 Ang Real-Space Cutoff, 9.0 Ang vdW Cutoff with Correction, 1.0 fs Time Step with Modified Beeman Integrator, and 0.00001 RMS Induced Dipole Convergence; OpenMP timings reported as "wall clock" time, with parallel speedup in parentheses

Machine Type (OS/Compiler)	CPU	Core/Thread	Seconds
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/1	5427.4 (1.00)
Mac Pro 1.1 (MacOS 10.11, GNU 7.1)	5150	4/4	2369.3 (2.29)
Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/1	4872.7 (1.00)

(continues on next page)

(continued from previous page)

Mac Pro 3.1 (MacOS 10.13, GNU 8.1)	E5462	8/8	1727.8 (2.82)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/1	3686.6 (1.00)
Mac Pro 5.1 (MacOS 10.13, GNU 8.1)	X5650	12/24	779.0 (4.73)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/1	2240.2 (1.00)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/2	1509.2 (1.00)
iMac 14.2 (MacOS 10.13, GNU 8.1)	4670	4/4	916.8 (1.00)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/1	2279.8 (1.00)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/2	1494.0 (1.00)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/4	897.6 (1.00)
MacBook Pro 11.3 (MacOS 10.13, GNU 8.1)	4960HQ	4/8	763.5 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/1	1621.2 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/2	1114.9 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/4	701.3 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/6	577.4 (1.00)
Razer Blade 15 (Ubuntu 18.04, GNU 7.5)	9750H	6/12	545.2 (1.00)

CHAPTER

THIRTEEN

ACKNOWLEDGMENTS

The TINKER package has developed over a period of many years, very slowly during the late-1980s, and more rapidly since the mid-1990s in Jay Ponder's research group at the Washington University School of Medicine in Saint Louis. Many people have played significant roles in the development of the package into its current form. The major contributors are listed below:

Stew Rubenstein coordinate interconversions; original optimization methods and torsional angle manipulation

Craig Kundrot molecular surface area & volume and their derivatives

Shawn Huston original AMBER/OPLS implementation; free energy calculations; time correlation functions

Mike Dudek initial multipole models for peptides and proteins

Yong "Mike" Kong multipole electrostatics; dipole polarization; reaction field treatment; TINKER water model

Reece Hart potential smoothing methodology; Scheraga's DEM, Straub's GDA and extensions

Mike Hodsdon extension of the TINKER distgeom program and its application to NMR NOE structure determination

Rohit Pappu potential smoothing methodology and PSS algorithms; rigid body optimization; GB/SA solvation derivatives

Wijnand Mooij MM3 directional hydrogen bonding term; crystal lattice minimization code

Gerald Loeffler stochastic/Langevin dynamics implementation

Marina Vorobieva nucleic acid building module and parameter translation Nina Sokolova

Peter Bagossi AMOEBA force field parameters for alkanes and diatomics

Pengyu Ren Ewald summation for polarizable atomic multipoles; AMOEBA force field for water, organics and peptides

Anders Carlsson original ligand field potential energy term for transition metals

Andrey Kutepov integrator for rigid-body dynamics trajectories

Tom Darden Particle Mesh Ewald (PME) code, and development of PME for the AMOEBA force field

Alan Grossfield Monte Carlo minimization; tophat potential smoothing

Michael Schnieders Force Field Explorer GUI for TINKER; neighbor lists for nonbonded interactions

Chuanjie Wu solvation free energy calculations; AMOEBA nucleic acid force field; parameterization tools for TINKER

Justin Xiang angular overlap and valence bond potential models for transition metals

David Gohara OpenMP parallelization of energy terms including PME, and parallel neighbor lists

It is critically important that TINKER's distributed force field parameter sets exactly reproduce the intent of the original force field authors. We would like to thank Julian Tirado-Rives (OPLS-AA), Alex MacKerell (CHARMM27), Wilfred van Gunsteren (GROMOS), and Adrian Roitberg and Carlos Simmerling (AMBER) for their help in testing TINKER's results against those given by the authentic programs and parameter sets. Lou Allinger provided updated parameters for MM2 and MM3 on several occasions. His very successful methods provided the original inspiration for the development of TINKER.

Still other workers have devoted considerable time in developing code that will hopefully be incorporated into future TINKER versions; for example, Jim Kress (UFF implementation) and Michael Sheets (numerous code optimizations, thermodynamic integration). Finally, we wish to thank the many users of the TINKER package for their suggestions and comments, praise and criticism, which have resulted in a variety of improvements.

CHAPTER

FOURTEEN

REFERENCES

This section contains a list of the references to general theory, algorithms and implementation details which have been of use during the development of the TINKER package. Methods described in some of the references have been implemented in detail within the TINKER source code. Other references contain useful background information although the algorithms themselves are now obsolete. Still other papers contain ideas or extensions planned for future inclusion in TINKER. References for specific force field parameter sets are provided in an earlier section of this User's Guide. This list is heavily skewed toward biomolecules in general and proteins in particular. This bias reflects our group's major interests; however an attempt has been made to include methods which should be generally applicable.

14.1 Partial List of Molecular Mechanics Software Packages

AMBER	Peter Kollman, University of California, San Francisco
AMMP	Rob Harrison, Thomas Jefferson University, Philadelphia
ARGOS	Andy McCammon, University of California, San Diego
BOSS	William Jorgensen, Yale University
BRUGEL	Shoshona Wodak, Free University of Brussels
CFF	Shneior Lifson, Weizmann Institute
CHARMM	Martin Karplus, Harvard University
CHARMM/GEMM	Bernard Brooks, National Institutes of Health, Bethesda
DELPHI	Bastian van de Graaf, Delft University of Technology
DISCOVER	Molecular Simulations Inc., San Diego
DL_POLY	W. Smith & T. Forester, CCP5, Daresbury Laboratory
ECEPP	Harold Scheraga, Cornell University
ENCAD	Michael Levitt, Stanford University
FANTOM	Werner Braun, University of Texas, Galveston
FEDER/2	Nobuhiro Go, Kyoto University
GROMACS	Herman Berendsen, University of Groningen
GROMOS	Wilfred van Gunsteren, BIOMOS and ETH, Zurich
IMPACT	Ronald Levy, Rutgers University
MACROMODEL	Schodinger, Inc., Jersey City, New Jersey
MM2/MM3/MM4	N. Lou Allinger, University of Georgia
MMC	Cliff Dykstra, Indiana Univ.-Purdue Univ. at Indianapolis
MMFF	Tom Halgren, Merck Research Laboratories, Rahway
MMTK	Konrad Hinsen, Inst. of Structural Biology, Grenoble
MOIL	Ron Elber, Cornell University

(continues on next page)

(continued from previous page)

MOLARIS	Arieh Warshal, University of Southern California
MOLDY	Keith Refson, Oxford University
MOSCITO	Dietmar Paschek & Alfons Geiger, Universitt Dortmund
NAMD	Klaus Schulten, University of Illinois, Urbana
OOMPAA	Andy McCammon, University of California, San Diego
ORAL	Karel Zimmerman, INRA, Jouy-en-Josas, France
ORIENT	Anthony Stone, Cambridge University
PCMODEL	Kevin Gilbert, Serena Software, Bloomington, Indiana
PEFF	Jan Dillen, University of Pretoria, South Africa
Q	Johan Aqvist, Uppsala University
SIBFA	Nohad Gresh, INSERM, CNRS, Paris
SIGMA	Jan Hermans, University of North Carolina
SPASIBA	Gerard Vergoten, UniversitÈ de Lille
SPASMS	David Spellmeyer and the Kollman Group, UCSF
TINKER	Jay Ponder, Washington University, St. Louis
XPLOR/CNS	Axel Brunger, Stanford University
YAMMP	Stephen Harvey, University of Alabama, Birmingham
YASP	Florian Mueller-Plathe, ETH Zentrum, Zurich
YETI	Angelo Vedani, Biografik-Labor 3R, Basel

AMBER D. A Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel and P. Kollman, AMBER, a Package of Computer Programs for Applying Molecular Mechanics, Normal Mode Analysis, Molecular Dynamics and Free Energy Calculations to Simulate the Structural and Energetic Properties of Molecules, Comp. Phys. Commun., 91, 1-41 (1995)

ARGOS T. P. Straatsma and J. A. McCammon, ARGOS, a Vectorized General Molecular Dynamics Program, J. Comput. Chem., 11, 943-951 (1990)

CHARMM 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. Comput. Chem., 4, 187-217 (1983)

ENCAD M. Levitt, M. Hirshberg, R. Sharon and V. Daggett, Potential Energy Function and Parameters for Simulations for the Molecular Dynamics of Proteins and Nucleic Acids in Solution, Comp. Phys. Commun., 91, 215-231 (1995)

FANTOM T. Schaumann, W. Braun and K. Wurtrich, The Program FANTOM for Energy Refinement of Polypeptides and Proteins Using a Newton-Raphson Minimizer in Torsion Angle Space, Biopolymers, 29, 679-694 (1990)

FEDER/2 H. Wako, S. Endo, K. Nagayama and N. Go, FEDER/2: Program for Static and Dynamic Conformational Energy Analysis of Macro-molecules in Dihedral Angle Space, Comp. Phys. Commun., 91, 233-251 (1995)

GROMACS E. Lindahl, B. Hess and D. van der Spoel, GROMACS 3.0: A Package for Molecular Simulation and Trajectory Analysis, J. Mol. Mol., 7, 306-317 (2001)

GROMOS W. R. P. Scott, P. H. Hunenberger , I. G. Tironi, A. E. Mark, S. R. Billeter, J. Fennen, A. E. Torda, T. Huber, P. Kruger, W. F. van Gunsteren, The GROMOS Biomolecular Simulation Program Package, J. Phys. Chem. A, 103, 3596-3607 (1999)

IMPACT D. B. Kitchen, F. Hirata, J. D. Westbrook, R. Levy, D. Kofke and M. Yarmush, Conserving Energy during Molecular Dynamics Simulations of Water, Proteins, and Proteins in Water, J. Comput. Chem., 10, 1169-1180 (1990)

MACROMODEL F. Mahamadi, N. G. J. Richards, W. C. Guida, R. Liskamp, M. Lipton, C. Caufield, G. Chang, T. Hendrickson and W. C. Still, MacroModel: An Integrated Software System for Modeling Organic and Bioorganic Molecules Using Molecular Mechanics, J. Comput. Chem., 11, 440-467 (1990)

MM2 N. L. Allinger, Conformational Analysis. 130. MM2. A Hydrocarbon Force Field Utilizing V1 and V2 Torsional Terms, J. Am. Chem. Soc., 99, 8127-8134 (1977)

MM3 N. L. Allinger, Y. H. Yuh and J.-H. Lii, Molecular Mechanics. The MM3 Force Field for Hydrocarbons, J. Am. Chem. Soc., 111, 8551-8566 (1989)

MM4 N. L. Allinger, K. Chen and J.-H. Lii, An Improved Force Field (MM4) for Saturated Hydrocarbons, J. Comput. Chem., 17, 642-668 (1996)

MMC C. E. Dykstra, Molecular Mechanics for Weakly Interacting Assemblies of Rare Gas Atoms and Small Molecules, J. Am. Chem. Soc., 111, 6168-6174 (1989)

MMFF T. A. Halgren, Merck Molecular Force Field. I. Basis, Form, Scope, Parameterization, and Performance of MMFF94, J. Comput. Chem., 17, 490-516 (1996)

MOIL R. Elber, A. Roitberg, C. Simmerling, R. Goldstein, H. Li, G. Verkhiver, C. Keasar, J. Zhang and A. Ulitsky, MOIL: A Program for Simulations of Macromolecules, Comp. Phys. Commun., 91, 159-189 (1995)

MOSCITO See the web site at http:/ganter.chemie.uni-dortmund.de/~pas/moscito.html

NAMD L. KalÈ, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan and K. Schulten, NAMD2: Greater Scalability for Parallel Molecular Dynamics, J. Comput. Phys., 151, 283-312 (1999)

OOMPAA G. A. Huber and J. A. McCammon, OOMPAA: Object-oriented Model for Probing Assemblages of Atoms, J. Comput. Phys., 151, 264-282 (1999)

ORAL K. Zimmermann, ORAL: All Purpose Molecular Mechanics Simulator and Energy Minimizer, J. Comput. Chem., 12, 310-319 (1991)

PCMODEL See the web site at http:/www.serenasoft.com

PEFF J. L. M. Dillen, PEFF: A Program for the Development of Empirical Force Fields, J. Comput. Chem., 13, 257-267 (1992)

Q See the web site at http://aqvist.bmc.uu.se/Q

SIBFA N. Gresh, Inter- and Intramolecular Interactions. Inception and Refinements of the SIBFA, Molecular Mechanics (SMM) Procedure, a Separable, Polarizable Methodology Grounded on ab Initio SCF/MP2 Computations. Examples of Applications to Molecular Recognition Problems, J. Chim. Phys. PCB, 94, 1365-1416 (1997)

SIGMA See the web site at http://femto.med.unc.edu/SIGMA

SPASIBA P. Derreumaux and G. Vergoten, A New Spectroscopic Molecular Mechanics Force-Field - Parameters For Proteins, J. Chem. Phys., 102, 8586-8605 (1995)

TINKER See the web site at http://dasher.wustl.edu/tinker

YAMMP R. K.-Z. Tan and S. C. Harvey, Yammp: Development of a Molecular Mechanics Program Using the Modular Programming Method, J. Comput. Chem., 14, 455-470 (1993)

YETI A. Vedani, YETI: An Interactive Molecular Mechanics Program for Small-Molecule Protein Complexes, J. Comput. Chem., 9, 269-280 (1988)

14.2 Molecular Mechanics

U. Burkert and N. L. Allinger, Molecular Mechanics, American Chemical Society, Washington, D.C., 1982

P. Comba and T. W. Hambley, Molecular Modeling of Inorganic Compounds, 2nd Ed., Wiley-VCH, New York, 2001

K. Machida, Principles of Molecular Mechanics, Kodansha/John Wiley & Sons, Tokyo/New York, 1999

A. K. Rappe and C. J. Casewit, Molecular Mechanics across Chemistry, University Science Books, Sausalito, CA, 1997

K. Rasmussen, Potential Energy Functions in Conformational Analysis (Lecture Notes in Chemistry, Vol. 27), Springer-Verlag, Berlin, 1985

14.3 Computer Simulation Methods

M. P. Allen and D. J. Tildesley, Computer Simulation of Liquids, Oxford University Press, Oxford, 1987

C. J. Cramer, Essentials of Computational Chemistry: Theories and Models, John Wiley and Sons, New York, 2002

M. J. Field, A Practical Introduction to the Simulation of Molecular Systems, Cambridge Univ. Press, Cambridge, 1999

D. Frankel and B. Smit, Understanding Molecular Simulation: From Algorithms to Applications, 2nd Ed., Academic Press, San Diego, CA, 2001

J. M. Haile, Molecular Dynamics Simulation: Elementary Methods, John Wiley and Sons, New York, 1992

F. Jensen, Introduction to Computational Chemistry, John Wiley and Sons, New York, 1998

A. R. Leach, Molecular Modelling: Principles and Applications, 2nd Ed., Addison Wesley Longman, Essex, England, 2001

D. C. Rapaport, The Art of Molecular Dynamics Simulation, 2nd Ed., Cambridge University Press, Cambridge, 2004

T. Schlick, Molecular Modeling and Simulation, Springer-Verlag, New York, 2002

14.4 Modeling of Biological Macromolecules

O. M. Becker, A. D. MacKerell, Jr., B. Roux and M. Watanabe, Eds., Computational Biochemistry and Biophysics, Marcel Dekker, New York, 2001

C. L. Brooks III, M. Karplus and B. M. Pettitt, Proteins: A Theoretical Perspective of Dynamics, Structure, and Thermodynamics, John Wiley and Sons, New York, 1988

V. Daggett, Ed., Protein Simulations (Advances in Protein Chemistry, Vol. 66), Academic Press/Elsevier, New York, 2003

J. A. McCammon and S. Harvey, Dynamics of Proteins and Nucleic Acids, Cambridge University Press, Cambridge, 1987

W. F. van Gunsteren, P. K. Weiner and A. J. Wilkinson, Computer Simulation of Biomolecular Systems, Vol. 1-3, Kluwer Academic Publishers, Dordrecht, 1989-1997

14.5 Conjugate Gradient and Quasi-Newton Optimization

J. Nocedal and S. J. Wright, Numerical Optimization, Springer-Verlag, New York, 1999

S. G. Nash and A. Sofer, Linear and Nonlinear Programming, McGraw-Hill, New York, 1996

R. Fletcher, Practical Methods of Optimization, John Wiley & Sons Ltd., Chichester, 1987

D. G. Luenberger, Linear and Nonlinear Programming, 2nd Ed., Addison-Wesley, Reading, MA, 1984

P. E. Gill, W. Murray and M. H. Wright, Practical Optimization, Academic Press, New York, 1981

J. Nocedal, Updating Quasi-Newton Matrices with Limited Storage, Math. Comp., 773-782 (1980)

S. J. Watowich, E. S. Meyer, R. Hagstrom and R. Josephs, A Stable, Rapidly Converging Conjugate Gradient Method for Energy Minimization, J. Comput. Chem., 9, 650-661 (1988)

W. C. Davidon, Optimally Conditioned Optimization Algorithms without Line Searches, Math. Prog., 9, 1-30 (1975)

14.6 Truncated Newton Optimization

J. W. Ponder and F. M. Richards, An Efficient Newton-like Method for Molecular Mechanics Energy Minimization of Large Molecules, J. Comput. Chem., 8, 1016-1024 (1987)

R. S. Dembo and T. Steihaug, Truncated-Newton Algorithms for Large-Scale Unconstrained Optimization, Math. Prog., 26, 190-212 (1983)

S. C. Eisenstat and H. F. Walker, Choosing the Forcing Terms in an Inexact Newton Method, SIAM J. Sci. Comput., 17, 16-32 (1996)

T. Schlick and M. Overton, A Powerful Truncated Newton Method for Potential Energy Minimization, J. Comput. Chem., 8, 1025-1039 (1987)

D. S. Kershaw, The Incomplete Cholesky-Conjugate Gradient Method for the Iterative Solution of Systems of Linear Equations, J. Comput. Phys., 26, 43-65 (1978)

T. A. Manteuffel, An Incomplete Factorization Technique for Positive Definite Linear Systems, Math. Comp., 34, 473-497 (1980)

P. Derreumaux, G. Zhang and T. Schlick and B. R. Brooks, A Truncated Newton Minimizer Adapted for CHARMM and Biomolecular Applications, J. Comput. Chem., 15, 532-552 (1994)

I. S. Duff, A. M. Erisman and J. K. Reid, Direct Methods for Sparse Matrices, Oxford University Press, Oxford, 1986

14.7 Potential Energy Smoothing

R. V. Pappu, R. K. Hart and J. W. Ponder, Analysis and Application of Potential Energy Smoothing Methods for Global Optimization, J. Phys. Chem. B, 102, 9725-9742 (1998)

L. Piela, J. Kostrowicki and H. A. Scheraga, The Multiple-Minima Problem in the Conformational Analysis of Molecules. Deformation of the Potential Energy Hypersurface by the Diffusion Equation Method, J. Phys. Chem., 93, 3339-3346 (1989)

J. Ma and J. E. Straub, Simulated Annealing Using the Classical Density Distribution, J. Chem. Phys., 101, 533-541 (1994)

C. Tsoo and C. L. Brooks, Cluster Structure Determination Using Gaussian Density Distribution Global Minimization Methods, J. Chem. Phys., 101, 6405-6411 (1994)

S. Nakamura, H. Hirose, M. Ikeguchi and J. Doi, Conformational Energy Minimization Using a Two-Stage Method, J. Phys. Chem., 99, 8374-8378 (1995)

T. Huber, A. E. Torda and W. F. van Gunsteren, Structure Optimization Combining Soft-Core Interaction Functions, the Diffusion Equation Method, and Molecular Dynamics, J. Phys. Chem. A, 101, 5926-5930 (1997)

S. Schelstraete and H. Verschelde, Finding Minimum-Energy Configurations of Lennard-Jones Clusters Using an Effective Potential, J. Phys. Chem. A, 101, 310-315 (1998)

I. Andricioaei and J. E. Straub, Global Optimization Using Bad Derivatives: Derivative-Free Method for Molecular Energy Minimization, J. Comput. Chem., 19, 1445-1455 (1998)

L. Piela, Search for the Most Stable Structures on Potential Energy Surfaces, Coll. Czech. Chem. Commun., 63, 1368-1380 (1998)

14.8 "Sniffer" Global Optimization

A. O. Griewank, Generalized Descent for Global Optimization, J. Opt. Theor. Appl., 34, 11-39 (1981)

R. A. R. Butler and E. E. Slaminka, An Evaluation of the Sniffer Global Optimization Algorithm Using Standard Test Functions, J. Comput. Phys., 99, 28-32 (1993)

J. W. Rogers and R. A. Donnelly, Potential Transformation Methods for Large-Scale Global Optimization, SIAM J. Optim., 5, 871-891 (1995)

14.9 Integration Methods for Molecular Dynamics

D. Beeman, Some Multistep Methods for Use in Molecular Dynamics Calculations, J. Comput. Phys., 20, 130-139 (1976)

M. Levitt and H. Meirovitch, Integrating the Equations of Motion, J. Mol. Biol., 168, 617-620 (1983)

J. Aqvist, W. F. van Gunsteren, M. Leijonmarck and O. Tapia, A Molecular Dynamics Study of the C-Terminal Fragment of the L7/L12 Ribosomal Protein, J. Mol. Biol., 183, 461-477 (1985)

W. C. Swope, H. C. Andersen, P. H. Berens and K. R. Wilson, A Computer Simulation Method for the Calculation of Equilibrium Constants for the Formation of Physical Clusters of Molecules: Application to Small Water Clusters, J. Chem. Phys., 76, 637-649 (1982)

14.10 Constraint Dynamics

W. F. van Gunsteren and H. J. C. Berendsen, Algorithms for Macromolecular Dynamics and Constraint Dynamics, Mol. Phys., 34, 1311-1327 (1977)

G. Ciccotti, M. Ferrario and J.-P. Ryckaert, Molecular Dynamics of Rigid Systems in Cartesian Coordinates: A General Formulation, Mol. Phys., 47, 1253-1264 (1982)

H. C. Andersen, Rattle: A "Velocity" Version of the Shake Algorithm for Molecular Dynamics Calculations, J. Comput. Phys., 52, 24-34 (1983)

R. Kutteh, RATTLE Recipe for General Holonomic Constraints: Angle and Torsion Constraints, CCP5 Newsletter, 46, 9-17 (1998) [available from the web site at http://www.dl.ac.uk/CCP/CCP5/newsletter_index.html]

B. J. Palmer, Direct Application of SHAKE to the Velocity Verlet Algorithm, J. Comput. Phys., 104, 470-472 (1993)

S. Miyamoto and P. A. Kollman, SETTLE: An Analytical Version of the SHAKE and RATTLE Algorithm for Rigid Water Models, J. Comput. Chem., 13, 952-962 (1992)

B. Hess, H. Bekker, H. J. C. Berendsen and J. G. E. M. Fraaije, LINCS: A Linear Constraint Solver for Molecular Simulations, J. Comput. Chem., 18, 1463-1472 (1997)

J. T. Slusher and P. T. Cummings, Non-Iterative Constraint Dynamics using Velocity-Explicit Verlet Methods, Mol. Simul., 18, 213-224 (1996)

14.11 Langevin, Brownian and Stochastic Dynamics

M. P. Allen, Brownian Dynamics Simulation of a Chemical Reaction in Solution, Mol. Phys., 40, 1073-1087 (1980)

W. F. van Gunsteren and H. J. C. Berendsen, Algorithms for Brownian Dynamics, Mol. Phys., 45, 637-647 (1982)

F. Guarnieri and W. C. Still, A Rapidly Convergent Simulation Method: Mixed Monte Carlo/Stochastic Dynamics, J. Comput. Chem., 15, 1302-1310 (1994)

M. G. Paterlini and D. M. Ferguson, Constant Temperature Simulations using the Langevin Equation with Velocity Verlet Integration, Chem. Phys., 236, 243-252 (1998)

14.12 Constant Temperature and Pressure Dynamics

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola and J. R. Haak, Molecular Dynamics with Coupling to an External Bath, J. Chem. Phys., 81, 3684-3690 (1984)

W. G. Hoover, Canonical Dynamics: Equilibrium Phase-space Distributions, Phys. Rev. A, 31, 1695-1697 (1985)

J. J. Morales, S. Toxvaerd and L. F. Rull, Computer Simulation of a Phase Transition at Constant Temperature and Pressure, Phys. Rev. A, 34, 1495-1498 (1986)

B. R. Brooks, Algorithms for Molecular Dynamics at Constant Temperature and Pressure, Internal Report of Division of Computer Research and Technology, National Institutes of Health, 1988.

M. Levitt, Molecular Dynamics of Native Protein: Computer Simulation of Trajectories, J. Mol. Biol., 168, 595-620 (1983)

14.13 Out-of-Plane Deformation Terms

J. R. Maple, U. Dinar and A. T. Hagler, Derivation of Force Fields for Molecular Mechanics and Dynamics from ab initio Energy Surfaces, Proc. Natl. Acad. Sci. USA, 85, 5350-5354 (1988)

S.-H. Lee, K. Palmo and S. Krimm, New Out-of-Plane Angle and Bond Angle Internal Coordinates and Related Potential Energy Functions for Molecular Mechanics and Dynamics Simulations, J. Comput. Chem., 20, 1067-1084 (1999)

14.14 Analytical Derivatives of Potential Functions

K. J. Miller, R. J. Hinde and J. Anderson, First and Second Derivative Matrix Elements for the Stretching, Bending, and Torsional Energy, J. Comput. Chem., 10, 63-76 (1989)

D. H. Faber and C. Altona, UTAH5: A Versatile Programme Package for the Calculation of Molecular Properties by Force Field Methods, Computers & Chemistry, 1, 203-213 (1977)

W. C. Swope and D. M. Ferguson, Alternative Expressions for Energies and Forces Due to Angle Bending and Torsional Energy, Report G320-3561, J. Comput. Chem., 13, 585-594 (1992)

A. Blondel and M. Karplus, New Formulation for Derivatives of Torsion Angles and Improper Torsion Angles in Molecular Mechanics: Elimination of Singularities, J. Comput. Chem., 17, 1132-1141 (1996)

R. E. Tuzun, D. W. Noid and B. G. Sumpter, Efficient Treatment of Out-of-Plane Bend and Improper Torsion Interactions in MM2, MM3, and MM4 Molecular Mechanics Calculations, J. Comput. Chem., 18, 1804-1811 (1997)

14.15 Torsional Space Derivatives and Normal Modes

M. Levitt, C. Sander and P. S. Stern, Protein Normal-mode Dynamics: Trypsin Inhibitor, Crambin, Ribonuclease and Lysozyme, J. Mol. Biol., 181, 423-447 (1985)

M. Levitt, Protein Folding by Restrained Energy Minimization and Molecular Dynamics, J. Mol. Biol., 170, 723-764 (1983)

H. Wako and N. Go, Algorithm for Rapid Calculation of Hessian of Conformational Energy Function of Proteins by Supercomputer, J. Comput. Chem., 8, 625-635 (1987)

H. Abe, W. Braun, T. Noguti and N. Go, Rapid Calculation of First and Second Derivatives of Conformational Energy with Respect to Dihedral Angles for Proteins: General Recurrent Equations, Computers & Chemistry, 8, 239-247 (1984)

T. Noguti and N. Go, A Method of Rapid Calculation of a Second Derivative Matrix of Conformational Energy for Large Molecules, J. Phys. Soc. Japan, 52, 3685-3690 (1983)

14.16 Analytical Surface Area and Volume

M. L. Connolly, Analytical Molecular Surface Calculation, J. Appl. Cryst., 16, 548-558 (1983)

M. L. Connolly, Computation of Molecular Volume, J. Am. Chem. Soc., 107, 1118-1124 (1985)

M. L. Connolly, Molecular Surfaces: A Review, available from the web site at http://www.netsci.org/Science/Compchem/feature14.html

C. E. Kundrot, J. W. Ponder and F. M. Richards, Algorithms for Calculating Excluded Volume and Its Derivatives as a Function of Molecular Conformation and Their Use in Energy Minimization, J. Comput. Chem., 12, 402-409 (1991)

T. J. Richmond, Solvent Accessible Surface Area and Excluded Volume in Proteins, J. Mol. Biol., 178, 63-89 (1984)

L. Wesson and D. Eisenberg, Atomic Solvation Parameters Applied to Molecular Dynamics of Proteins in Solution, Protein Science, 1, 227-235 (1992)

V. Gononea and E. Osawa, Implementation of Solvent Effect in Molecular Mechanics, Part 3. The First- and Second-order Analytical Derivatives of Excluded Volume, J. Mol. Struct. (Theochem), 311 305-324 (1994)

K. D. Gibson and H. A. Scheraga, Exact Calculation of the Volume and Surface Area of Fused Hard-sphere Molecules with Unequal Atomic Radii, Mol. Phys., 62, 1247-1265 (1987)

K. D. Gibson and H. A. Scheraga, Surface Area of the Intersection of Three Spheres with Unequal Radii: A Simplified Analytical Formula, Mol. Phys., 64, 641-644 (1988)

S. Sridharan, A. Nichols and K. A. Sharp, A Rapid Method for Calculating Derivatives of Solvent Accessible Surface Areas of Molecules, J. Comput, Chem., 16, 1038-1044 (1995)

14.17 Approximate Surface Area and Volume

S. J. Wodak and J. Janin, Analytical Approximation to the Accessible Surface Area of Proteins, Proc. Natl. Acad. Sci. USA, 77, 1736-1740 (1980)

W. Hasel, T. F. Hendrickson and W. C. Still, A Rapid Approximation to the Solvent Accessible Surface Areas of Atoms, Tetrahedron Comput. Method., 1, 103-116 (1988)

J. Weiser, P. S. Shenkin and W. C. Still, Approximate Solvent-Accessible Surface Areas from Tetrahedrally Directed Neighber Densities, Biopolymers, 50, 373-380 (1999)

14.18 Boundary Conditions and Neighbor Methods

W. F. van Gunsteren, H. J. C. Berendsen, F. Colonna, D. Perahia, J. P. Hollenberg and D. Lellouch, On Searching Neighbors in Computer Simulations of Macromolecular Systems, J. Comput. Chem., 5, 272-279 (1984)

F. Sullivan, R. D. Mountain and J. O'Connell, Molecular Dynamics on Vector Computers, J. Comput. Phys., 61, 138-153 (1985)

J. Boris, A Vectorized "Near Neighbors" Algorithm of Order N Using a Monotonic Logical Grid, J. Comput. Phys., 66, 1-20 (1986)

S. G. Lambrakos and J. P. Boris, Geometric Properties of the Monotonic Lagrangian Grid Algorithm for Near Neighbors Calculations, J. Comput. Phys., 73, 183-202 (1987)

T. A. Andrea, W. C. Swope and H. C. Andersen, The Role of Long Ranged Forces in Determining the Structure and Properties of Liquid Water, J. Chem. Phys., 79, 4576-4584 (1983)

D. N. Theodorou and U. W. Suter, Geometrical Considerations in Model Systems with Periodic Boundary Conditions, J. Chem. Phys., 82, 955-966 (1985)

J. Barnes and P. Hut, A Hierarchical O(NlogN) Force-calculation Algorithm, Nature, 234, 446-449 (1986)

14.19 Cutoff and Truncation Methods

P. J. Steinbach and B. R. Brooks, New Spherical-Cutoff Methods for Long-Range Forces in Macromolecular Simulation, J. Comput. Chem., 15, 667-683 (1993)

R. J. Loncharich and B. R. Brooks, The Effects of Truncating Long-Range Forces on Protein Dynamics, Proteins, 6, 32-45 (1989)

C. L. Brooks III, B. M. Pettitt and M. Karplus, Structural and Energetic Effects of Truncating Long Ranged Interactions in Ionic and Polar Fluids, J. Chem. Phys., 83, 5897-5908 (1985)

14.20 Ewald Summation Techniques

A. Y. Toukmaji and J. A. Board, Jr., Ewald Summation Techniques in Perspective: A Survey, Comp. Phys. Commun., 95, 73-92 (1996)

T. Darden, L. Perera, L. Li and L. Pedersen, New Tricks for Modelers from the Crystallography Toolkit: The Particle Mesh Ewald Algorithm and its Use in Nucleic Acid Simulations, Structure, 7, R550-R60 (1999)

T. Darden, D. York and L. G. Pedersen, Particle Mesh Ewald: An Nlog(N) Method for Ewald Sums in Large Systems, J. Chem. Phys., 98, 10089-10092 (1993)

U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee and L. G. Pedersen, A Smooth Particle Mesh Ewald Method, J. Chem. Phys., 103, 8577-8593 (1995)

W. Smith, Point Multipoles in the Ewald Summation (Revisited), CCP5 Newsletter, 46, 18-30 (1998) [available from http://www.dl.ac.uk/CCP/CCP5/newsletter_index.html]

S. E. Feller, R. W. Pastor, A. Rojnuckarin, S. Bogusz and B. R. Brooks, Effect of Electrostatic Force Truncation on Interfacial and Transport Properties of Water, J. Phys. Chem., 100, 17011-17020 (1996)

W. Weber, P. H. H¸nenberger and J. A. McCammon, Molecular Dynamics Simulations of a Polyalanine Octapeptide under Ewald Boundary Conditions: Influence of Artificial Periodicity on Peptide Conformation, J. Phys. Chem. B, 104, 3668-3675 (2000)

14.21 Conjugated and Aromatic Systems

N. L. Allinger, F. Li, L. Yan and J. C. Tai, Molecular Mechanics (MM3) Calculations on Conjugated Hydrocarbons, J. Comput. Chem., 11, 868-895 (1990)

J. T. Sprague, J. C. Tai, Y. Yuh and N. L. Allinger, The MMP2 Calculational Method, J. Comput. Chem., 8, 581-603 (1987)

J. Kao, A Molecular Orbital Based Molecular Mechanics Approach to Study Conjugated Hydrocarbons, J. Am. Chem. Soc., 109, 3818-3829 (1987)

J. Kao and N. L. Allinger, Conformational Analysis: Heats of Formation of Conjugated Hydrocarbons by the Force Field Method, J. Am. Chem. Soc., 99, 975-986 (1977)

D. H. Lo and M. A. Whitehead, Accurate Heats of Atomization and Accurate Bond Lengths: Benzenoid Hydrocarbons, Can. J. Chem., 46, 2027-2040 (1968)

G. D. Zeiss and M. A. Whitehead, Hetero-atomic Molecules: Semi-empirical Molecular Orbital Calculations and Prediction of Physical Properties, J. Chem. Soc. A, 1727-1738 (1971)

14.22 Free Energy Simulation Methods

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

B. L. Tembe and J. A. McCammon, Ligand-Receptor Interactions, Computers & Chemistry, 8, 281-283 (1984)

W. L. Jorgensen and C. Ravimohan, Monte Carlo Simulation of Differences in Free Energy of Hydration, J. Chem. Phys., 83, 3050-3054 (1985)

W. L. Jorgensen, J. K. Buckner, S. Boudon and J. Tirado-Rives, Efficient Computation of Absolute Free Energies of Binding by Computer Simulations: Application to the Methane Dimer in Water, J. Chem. Phys., 89, 3742-3746 (1988)

S. H. Fleischman and C. L. Brooks III, Thermodynamics of Aqueous Solvation: Solution Properties of Alcohols and Alkanes, J. Chem. Phys., 87, 3029-3037 (1987)

U. C. Singh, F. K. Brown, P. A. Bash and P. A. Kollman, An Approach to the Application of Free Energy Perturbation Methods Using Molecular Dynamics, J. Am. Chem. Soc., 109, 1607-1614 (1987)

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

T. P. Straatsma, H. J. C. Berendsen and J. P. M. Postma, Free Energy of Hydrophobic Hydration: A Molecular Dynamics Study of Noble Gases in Water, J. Chem. Phys., 85, 6720-6727 (1986)

T. P. Straatsma and H. J. C. Berendsen, Free Energy of Ionic Hydration: Analysis of a Thermodynamic Integration Technique to Evaluate Free Energy Differences by Molecular Dynamics Simulations, J. Chem. Phys., 89, 5876-5886 (1988)

M. Mezei, The Finite Difference Thermodynamic Integration, Tested on Calculating the Hydration Free Energy Difference between Acetone and Dimethylamine in Water, J. Chem. Phys., 86, 7084-7088 (1987)

A. E. Mark and W. F. van Gunsteren, Decomposition of the Free Energy of a System in Terms of Specific Interactions, J. Mol. Biol., 240, 167-176 (1994)

S. Boresch and M. Karplus, The Meaning of Copmponent Analysis: Decomposition of the Free Energy in Terms of Specific Interactions, J. Mol. Biol., 254, 801-807 (1995)

14.23 Methods for Parameter Determination

N. L. Allinger, X. Zhou and J. Bergsma, Molecular Mechanics Parameters, J. Mol. Struct. (THEOCHEM), 312, 69-83 (1994)

A. J. Pertsin and A. I. Kitaigorodsky, The Atom-Atom Potential Method: Application to Organic Molecular Solids, Springer-Verlag, Berlin, 1987

D. E. Williams, Transferable Empirical Nonbonded Potential Functions, in Crystal Cohesion and Conformational Energies, Ed. by R. M. Metzger, Springer-Verlag, Berlin, 1981

A. T. Hagler and S. Lifson, A Procedure for Obtaining Energy Parameters from Crystal Packing, Acta Cryst., B30, 1336-1341 (1974)

A. T. Hagler, S. Lifson and P. Dauber, Consistent Force Field Studies of Intermolecular Forces in Hydrogen-Bonded Crystals: A Benchmark for the Objective Comparison of Alternative Force Fields, J. Am. Chem. Soc., 101, 5122-5130 (1979)

W. L. Jorgensen, J. D. Madura and C. J. Swenson, Optimized Intermolecular Potential Functions for Liquid Hydrocarbons, J. Am. Chem. Soc., 106, 6638-6646 (1984)

W. L. Jorgensen and C. J. Swenson, Optimized Intermolecular Potential Functions for Amides and Peptides: Structure and Properties of Liquid Amides, J. Am. Chem. Soc., 107, 569-578 (1985)

J. R. Maple, U. Dinur and A. T. Hagler, Derivation of Force Fields for Molecular Mechanics and Dynamics from ab Initio Surfaces, Proc. Nat. Acad. Sci. USA, 85, 5350-5354 (1988)

U. Dinur and A. T. Hagler, Direct Evaluation of Nonbonding Interactions from ab Initio Calculations, J. Am. Chem. Soc., 111, 5149-5151 (1989)

14.24 Electrostatic Interactions

S. L. Price, Towards More Accurate Model Intermolecular Potentials for Organic Molecules, Rev. Comput. Chem., 14, 225-289 (2000)

C. H. Faerman and S. L. Price, A Transferable Distributed Multipole Model for the Electrostatic Interactions of Peptides and Amides, J. Am. Chem. Soc., 112, 4915-4926 (1990)

C. E. Dykstra, Electrostatic Interaction Potentials in Molecular Force Fields, Chem. Rev., 93, 2339-2353 (1993)

M. J. Dudek and J. W. Ponder, Accurate Modeling of the Intramolecular Electrostatic Energy of Proteins, J. Comput. Chem., 16, 791-816 (1995)

U. Koch and E. Egert, An Improved Description of the Molecular Charge Density in Force Fields with Atomic Multipole Moments, J. Comput. Chem., 16, 937-944 (1995)

D. E. Williams, Representation of the Molecular Electrostatic Potential by Atomic Multipole and Bond Dipole Models, J. Comput. Chem., 9, 745-763 (1988)

F. Colonna, E. Evleth and J. G. Angyan, Critical Analysis of Electric Field Modeling: Formamide, J. Comput. Chem., 13, 1234-1245 (1992)

14.25 Polarization Effects

S. Kuwajima and A. Warshel, Incorporating Electric Polarizabilities in Water-Water Interaction Potentials, J. Phys. Chem., 94, 460-466 (1990)

J. W. Caldwell and P. A. Kollman, Structure and Properties of Neat Liquids Using Nonadditive Molecular Dynamics: Water, Methanol, and N-Methylacetamide, J. Phys. Chem., 99, 6208-6219 (1995)

D. N. Bernardo, Y. Ding, K. Kroegh-Jespersen and R. M. Levy, An Anisotropic Polarizable Water Model: Incorporation of All-Atom Polarizabilities into Molecular Mechanics Force Fields, J. Phys. Chem., 98, 4180-4187 (1994)

P. T. van Duijnen and M. Swart, Molecular and Atomic Polarizabilities: Thole's Model Revisited, J. Phys. Chem. A, 102, 2399-2407 (1998)

K. J. Miller, Calculation of the Molecular Polarizability Tensor, J. Am. Chem. Soc., 112, 8543-8551 (1990)

J. Applequist, J. R. Carl and K.-K. Fung, An Atom Dipole Interaction Model for Molecular Polarizability. Application to Polyatomic Molecules and Determination of Atom Polarizabilities, J. Am. Chem. Soc., 94, 2952-2960 (1972)

J. Applequist, Atom Charge Transfer in Molecular Polarizabilities. Application of the Olson-Sundberg Model to Aliphatic and Aromatic Hydrocarbons, J. Phys. Chem., 97, 6016-6023 (1993)

A. J. Stone, Distributed Polarizabilities, Mol. Phys., 56, 1065-1082 (1985)

J. M. Stout and C. E. Dykstra, A Distributed Model of the Electrical Response of Organic Molecules, J. Phys. Chem. A, 102, 1576-1582 (1998)

14.26 Macroscopic Treatment of Solvent

C. J. Cramer and D. G. Truhlar, Continuum Solvation Models: Classical and Quantum Mechanical Implementations, Rev. Comput. Chem., 6, 1-72 (1995)

B. Roux and T. Simonson, Implicit Solvation Models, Biophys. Chem., 78, 1-20 (1999)

M. K. Gilson, Introduction to Continuum Electrostatics with Molecular Applications, available from http://gilsonlab.umbi.umd.edu

14.27 Surface Area-Based Solvation Models

D. Eisenberg and A. D. McLachlan, Solvation Energy in Protein Folding and Binding, Nature, 319, 199-203 (1986)

L. Wesson and D. Eisenberg, Atomic Solvation Parameters Applied to Molecular Dynamics of Proteins in Solution, Prot. Sci., 1, 227-235 (1992)

T. Ooi, M. Oobatake, G. Nemethy and H. A. Scheraga, Accessible Surface Areas as a Measure of the Thermodynamic Parameters of Hydration of Peptides, Proc. Natl. Acad. Sci. USA, 84, 3086-3090 (1987)

J. D. Augspurger and H. A. Scheraga, An Efficient, Differentiable Hydration Potential for Peptides and Proteins, J. Comput. Chem., 17, 1549-1558 (1996)

14.28 Generalized Born Solvation Models

W. C. Still, A. Tempczyk, R. C. Hawley and T. Hendrickson, A Semiempirical Treatment of Solvation for Molecular Mechanics and Dynamics, J. Am. Chem. Soc., 112, 6127-6129 (1990)

D. Qiu, P. S. Shenkin, F. P. Hollinger and W. C. Still, The GB/SA Continuum Model for Solvation. A Fast Analytical Method for the Calculation of Approximate Born Radii, J. Phys. Chem. A, 101, 3005-3014 (1997)

G. D. Hawkins, C. J. Cramer and D. G. Truhlar, Pairwise Solute Descreening of Solute Charges from a Dielectric Medium, Chem. Phys. Lett., 246, 122-129 (1995)

G. D. Hawkins, C. J. Cramer and D. G. Truhlar, Parametrized Models of Aqueous Free Energies of Solvation Based on Pairwise Descreening of Solute Atomic Charges from a Dielectric Medium, J. Phys. Chem., 100, 19824-19839 (1996)

A. Onufriev, D. Bashford and D. A. Case, Modification of the Generalized Born Model Suitable for Macromolecules, J. Phys. Chem. B, 104, 3712-3720 (2000)

M. Schaefer and M. Karplus, A Comprehensive Analytical Treatment of Continuum Electrostatics, J. Phys. Chem., 100, 1578-1599 (1996)

M. Schaefer, C. Bartels and M. Karplus, Solution Conformations and Thermodynamics of Structured Peptides: Molecular Dynamics Simulation with an Implicit Solvation Model, J. Mol. Biol., 284, 835-848 (1998)

14.29 Superposition of Coordinate Sets

S. J. Kearsley, An Algorithm for the Simultaneous Superposition of a Structural Series, J. Comput. Chem., 11, 1187-1192 (1990)

R. Diamond, A Note on the Rotational Superposition Problem, Acta Cryst., A44, 211-216 (1988)

A. D. McLachlan, Rapid Comparison of Protein Structures, Acta Cryst., A38, 871-873 (1982)

S. C. Nyburg, Some Uses of a Best Molecular Fit Routine, Acta Cryst., B30, 251-253 (1974)

14.30 Location of Transition States

R. Czerminski and R. Elber, Reaction Path Study of Conformational Transitions and Helix Formation in a Tetrapeptide, Proc. Nat. Acad. Sci. USA, 86, 6963 (1989)

R. S. Berry, H. L. Davis and T. L. Beck, Finding Saddles on Multidimensional Potential Surfaces, Chem. Phys. Lett., 147, 13 (1988)

K. Muller, Reaction Paths on Multidimensional Energy Hypersurfaces, Ang. Chem. Int. Ed. Engl., 19, 1-13 (1980)

S. Bell and J. S. Crighton, Locating Transition States, J. Chem. Phys., 80, 2464-2475 (1984)

S. Fischer and M. Karplus, Conjugate Peak Refinement: An Algorithm for Finding Reaction Paths and Accurate Transition States in Systems with Many Degrees of Freedom, Chem. Phys. Lett., 194, 252-261 (1992)

J. E. Sinclair and R. Fletcher, A New Method of Saddle-Point Location for the Calculation of Defect Migration Energies, J. Phys. C, 7, 864-870 (1974)

R. Elber and M. Karplus, A Method for Determining Reaction Paths in Large Molecules: Application to Myoglobin, Chem. Phys. Lett., 139, 375-380 (1987)

D. T. Nguyen and D. A. Case, On Finding Stationary States on Large-Molecule Potential Energy Surfaces, J. Phys. Chem., 89, 4020-4026 (1985)

T. A. Halgren and W. N. Lipscomb, The Synchronous-Transit Method for Determining Reaction Pathways and Locating Molecular Transition States, Chem. Phys. Lett., 49, 225-232 (1977)

G. T. Barkema and N. Mousseau, Event-Based Relaxation of Continuous Disordered Systems, Phys. Rev. Lett., 77, 4358-4361 (1996)