|  |  |
| --- | --- |
| **1.** | **Descriptions of TINKER Routines** |

The distribution version of the TINKER package contains over 700 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

**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

**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

**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

**ANALYZ4 Subroutine**

"analyz4" prints the energy to 4 decimal places and number of interactions for each component of the potential energy

**ANALYZ6 Subroutine**

"analyz6" prints the energy to 6 decimal places and number of interactions for each component of the potential energy

**ANALYZ8 Subroutine**

"analyz8" prints the energy to 8 decimal places and number of interactions for each component of the potential energy

**ANALYZE Program**

"analyze" computes and displays the total potential; options are provided to partition the energy by atom or by potential function type; parameters used in computing interactions can also be displayed by atom; output of large energy interactions and of electrostatic and inertial properties is available

**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

**ARCHIVE Program**

"archive" is a utility program for coordinate files which concatenates multiple coordinate sets into a single archive file, or extracts individual coordinate sets from an 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)

**BASEFILE Subroutine**

"basefile" extracts from an input filename the portion consisting of any directory name and the base filename

**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 Hessain along the directions of a 2D spline grid

**BEEMAN Subroutine**

"beeman" performs a single molecular dynamics time step by means of a Beeman multistep recursion formula; the actual coefficients are 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 block of repeated units

**BITORS Subroutine**

"bitors" finds the total number of bitorsions, pairs of overlapping dihedral 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

**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 GB/SA 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

**BSET Subroutine**

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

**BSPLINE Subroutine**

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

**BSPLINE1 Subroutine**

"bspline1" calculates the coefficients and derivative coefficients for an n-th order B-spline approximation

**BSSTEP Subroutine**

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

**CALENDAR Subroutine**

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

**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 the array "wsave" which is used in both forward and backward transforms; the prime factorization of "n" together with a tabulation of the trigonometric functions are computed and stored in "wsave"

**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

**CHKCLASH Subroutine**

"chkclash" determines if there are any atom clashes which might cause trouble on subsequent energy evaluation

**CHKPOLE Subroutine**

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

**CHKRING Subroutine**

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

**CHKSIZE Subroutine**

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

**CHKTREE Subroutine**

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

**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 assumed to be a real symmetric positive definite matrix with its diagonal and upper triangle stored by rows

**CIRPLN Subroutine**

**CJKM Function**

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

**CLIMBER Subroutine**

**CLIMBRGD Subroutine**

**CLIMBROT Subroutine**

**CLIMBTOR Subroutine**

**CLIMBXYZ Subroutine**

**CLOCK Subroutine**

"clock" determines elapsed CPU time in seconds since the start of the job

**CLUSTER Subroutine**

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

**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

**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 program which converts between fractional and Cartesian coordinates, and can generate full unit cells from asymmetric units

**CUTOFFS Subroutine**

"cutoffs" initializes and stores spherical energy cutoff distance windows, Hessian element and Ewald sum cutoffs, and the pairwise neighbor generation method

**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

**DELETE Subroutine**

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

**DEPTH Function**

**DESTROYJVM Subroutine**

**DESTROYSERVER Subroutine**

**DFTMOD Subroutine**

"dftmod" computes the modulus of the discrete Fourier transform of "bsarray", storing it into "bsmod"

**DIAGQ Subroutine**

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

**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 dynamics trajectory in any of several statistical mechanical ensembles with optional periodic boundaries and optional coupling to temperature and pressure baths alternatively a stochastic dynamics trajectory can be generated

**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 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 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 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 or Fourier angle bending terms are optionally used

**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 to locate neighboring atoms

**EBUCK0C Subroutine**

"ebuck0c" 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 to locate neighboring atoms

**EBUCK1C Subroutine**

"ebuck1c" 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 to locate neighboring atoms

**EBUCK3C Subroutine**

"ebuck3c" 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 to locate neighboring atoms

**ECHARGE0C Subroutine**

"echarge0c" calculates the charge-charge interaction energy for use with potential smoothing methods

**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 to locate neighboring atoms

**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 to locate neighboring atoms

**ECHARGE1C Subroutine**

"echarge1c" calculates the charge-charge interaction energy and first derivatives with respect to Cartesian coordinates for use with potential smoothing methods

**ECHARGE1D Subroutine**

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

**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 double loop

**ECHARGE2B Subroutine**

"echarge2b" calculates second derivatives of the charge-charge interaction energy for a single atom for use with potential smoothing methods

**ECHARGE2C Subroutine**

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

**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 to locate neighboring atoms

**ECHARGE3C Subroutine**

"echarge3c" calculates the charge-charge interaction energy and partitions the energy among the atoms for use with potential smoothing methods

**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 to locate neighboring atoms

**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

**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

**EGAUSS Subroutine**

"egauss" calculates the Gaussian expansion van der Waals interaction energy

**EGAUSS0A Subroutine**

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

**EGAUSS0B Subroutine**

"egauss0b" calculates the Gaussian expansion van der Waals interaction 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 interaction energy and its first derivatives for use with stophat 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 stophat 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 interaction energy and partitions the energy among the atoms using a pairwise double loop

**EGAUSS3B Subroutine**

"egauss3b" calculates the Gaussian expansion van der Waals interaction energy and partitions the energy among the atoms using a pairwise double loop

**EGBSA0A Subroutine**

"egbsa0a" calculates the generalized Born polarization energy for the GB/SA solvation models

**EGBSA0B Subroutine**

"egbsa0b" 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

**EGBSA1A Subroutine**

"egbsa1a" calculates the generalized Born energy and first derivatives of the GB/SA solvation models

**EGBSA1B Subroutine**

"egbsa1b" calculates the generalized Born energy and first derivatives of the GB/SA solvation models for use with potential smoothing methods

**EGBSA2A Subroutine**

"egbsa2a" calculates second derivatives of the generalized Born energy term for the GB/SA solvation models

**EGBSA2B Subroutine**

"egbsa2b" calculates second derivatives of the generalized Born energy term for the GB/SA solvation models for use with potential smoothing methods

**EGBSA3A Subroutine**

"egbsa3a" calculates the generalized Born energy term for the GB/SA solvation models; also partitions the energy among the atoms

**EGBSA3B Subroutine**

"egbsa3b" 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; 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

**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**

"ehal0a" calculates the buffered 14-7 van der Waals energy using the method of lights to locate neighboring atoms

**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 to locate neighboring atoms

**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 to locate neighboring 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 to locate neighboring atoms

**ELJ0C Subroutine**

"elj0c" calculates the Lennard-Jones 6-12 van der Waals energy via a Gaussian approximation for potential energy smoothing

**ELJ0D Subroutine**

"elj0d" 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 to locate neighboring atoms

**ELJ1C Subroutine**

"elj1c" calculates the Lennard-Jones 6-12 van der Waals energy and its first derivatives via a Gaussian approximation for potential energy smoothing

**ELJ1D Subroutine**

"elj1d" 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 to locate neighboring atoms

**ELJ3C Subroutine**

"elj3c" 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

**ELJ3D Subroutine**

"elj3d" 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 to locate neighboring atoms

**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 to locate neighboring atoms

**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 to locate neighboring atoms

**EMPOLE Subroutine**

"empole" calculates the electrostatic energy due to atomic multipole interactions and dipole polarizability

**EMPOLE0A Subroutine**

"empole0a" calculates the electrostatic energy due to atomic multipole interactions and dipole polarizability using a pairwise double loop

**EMPOLE0B Subroutine**

"empole0b" calculates the electrostatic energy due to atomic multipole interactions and dipole polarizability using a regular Ewald summation

**EMPOLE1 Subroutine**

"empole1" calculates the multipole and dipole polarization energy and derivatives with respect to Cartesian coordinates

**EMPOLE1A Subroutine**

"empole1a" calculates the multipole and dipole polarization energy and derivatives with respect to Cartesian coordinates using a pairwise double loop

**EMPOLE1B Subroutine**

"empole1b" calculates the multipole and dipole polarization energy and derivatives with respect to Cartesian coordinates using a regular Ewald summation

**EMPOLE2 Subroutine**

"empole2" calculates second derivatives of the multipole and dipole polarization energy for a single atom at a time

**EMPOLE2A Subroutine**

"empole2a" computes multipole and dipole polarization first derivatives for a single atom with respect to Cartesian coordinates; used to get finite difference second derivatives

**EMPOLE3 Subroutine**

"empole3" calculates the electrostatic energy due to atomic multipole interactions and dipole polarizability, and partitions the energy among the atoms

**EMPOLE3A Subroutine**

"empole3a" calculates the electrostatic energy due to atomic multipole interactions and dipole polarizability, and partitions the energy among the atoms using a double loop

**EMPOLE3B Subroutine**

"empole3b" calculates the electrostatic energy due to atomic multipole interactions and dipole polarizability, and partitions the energy among the atoms using a regular Ewald summation

**ENERGY Function**

"energy" calls the subroutines to calculate the potential energy terms and sums up to form the total energy

**ENRGYZE Subroutine**

"energyze" 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 angle bend

**EOPBEND1 Subroutine**

"eopbend1" computes the out-of-plane bend potential energy and first derivatives at trigonal centers via a Wilson-Decius-Cross angle bend

**EOPBEND2 Subroutine**

"eopbend2" calculates second derivatives of the out-of-plane bend energy via a Wilson-Decius-Cross angle bend for a single atom using finite difference methods

**EOPBEND2A Subroutine**

"eopbend2a" calculates out-of-plane bending first derivatives at a trigonal center via a Wilson-Decius-Cross angle bend; 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 angle bend; 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

**EPITORS Subroutine**

"epitors" calculates the pi-orbital torsion potential energy

**EPITORS1 Subroutine**

"epitors1" calculates the pi-orbital torsion potential energy and first derivatives with respect to Cartesian coordinates

**EPITORS2 Subroutine**

"epitors2" calculates the second derivatives of the pi-orbital torsion energy for a single atom using finite difference methods

**EPITORS2A Subroutine**

"epitors2a" calculates the pi-orbital torsion first derivatives; used in computation of finite difference second derivatives

**EPITORS3 Subroutine**

"epitors3" calculates the pi-orbital torsion potential energy; also partitions the energy terms among the atoms

**EPME Subroutine**

"epme" computes the reciprocal space energy for a particle mesh Ewald summation over partial charges

**EPME1 Subroutine**

"epme1" computes the reciprocal space energy and first derivatives for a particle mesh Ewald summation

**EPME3 Subroutine**

"epme3" computes the reciprocal space energy for a particle mesh Ewald summation over partial charges and prints information about the energy over the charge grid points

**EPUCLC Subroutine**

**EREAL Subroutine**

"ereal" evaluates the real space portion of the regular Ewald summation energy due to atomic multipole interactions and dipole polarizability

**EREAL1 Subroutine**

"ereal1" evaluates the real space portion of the regular Ewald summation energy and gradient due to atomic multipole interactions and dipole polarizability

**EREAL3 Subroutine**

"ereal3" evaluates the real space portion of the regular Ewald summation energy due to atomic multipole interactions and dipole polarizability and partitions the energy among the atoms

**ERECIP Subroutine**

"erecip" evaluates the reciprocal space portion of the regular Ewald summation energy due to atomic multipole interactions and dipole polarizability

**ERECIP1 Subroutine**

"erecip1" evaluates the reciprocal space portion of the regular Ewald summation energy and gradient due to atomic multipole interactions and dipole polarizability

**ERECIP3 Subroutine**

"erecip3" evaluates the reciprocal space portion of the regular Ewald summation energy due to atomic multipole interactions and dipole polarizability, and prints information about the energy over the reciprocal lattice vectors

**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 erf for a real 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 continuum solvation energy via either the Eisenberg-McLachlan ASP model, Ooi-Scheraga SASA model, various GB/SA methods or the ACE model

**ESOLV1 Subroutine**

"esolv1" calculates the continuum solvation energy and first derivatives with respect to Cartesian coordinates using either the Eisenberg-McLachlan ASP, Ooi-Scheraga SASA or various GB/SA solvation models

**ESOLV2 Subroutine**

"esolv2" calculates second derivatives of the continuum solvation energy using either the Eisenberg-McLachlan ASP, Ooi-Scheraga SASA or various GB/SA solvation models

**ESOLV3 Subroutine**

"esolv3" calculates the continuum solvation energy using either the Eisenberg-McLachlan ASP model, Ooi-Scheraga SASA model, various GB/SA methods or the ACE model; 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

**EWALDCOF Subroutine**

"ewaldcof" finds a value of the Ewald coefficient such that all terms beyond the specified cutoff distance will have an value less than a specified tolerance

**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

**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**

**FFTFRONT Subroutine**

**FFTSETUP Subroutine**

**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, 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

**FIXPDB Subroutine**

"fixpdb" corrects problems with PDB files by converting residue and atom names to the forms used by TINKER

**FRACDIST Subroutine**

"fracdist" computes a normalized distribution of the pairwise fractional distances between the smoothed upper and lower bounds

**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]

**GDA Program**

"gda" implements Gaussian Density Annealing (GDA) algorithm for global optimization via simulated annealing

**GDA1 Subroutine**

**GDA2 Function**

**GDA3 Subroutine**

**GDASTAT Subroutine**

**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

**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

**GETIME Subroutine**

"getime" gets elapsed CPU time in seconds for an interval

**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

**GETMOL2 Subroutine**

"getmol2" asks for a Sybyl 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" searchs 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" searchs for a quoted text string within an input character string; the region between the first and second quotes is returned as the "text"; if the actual text is too long, only the first part is returned

**GETTEXT Subroutine**

"gettext" searchs an input string for the first string of non-blank characters; the region from a non-blank character to the first blank space is returned as "text"; if the actual text is too long, only the first part is returned

**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" searchs 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 comma 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

**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

**GRAFIC Subroutine**

"grafic" outputs the upper & lower triangles and diagonal of a square matrix in a schematic form for visual inspection

**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

**GYRATE Subroutine**

"gyrate" computes the radius of gyration of a molecular system from its atomic coordinates

**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

**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 the numerical Hessian elements with respect to torsional angles; either the full matrix or just the diagonal can be calculated; the full matrix needs nomega+1 gradient evaluations while the diagonal requires just two gradient calls

**HIMPTOR Subroutine**

"himptor" constructs hybrid improper torsional parameters given an initial state, final state and "lambda" value

**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 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"

**INDUCE Subroutine**

"induce" computes the induced dipole moment at each polarizable site due to direct or mutual polarization; assumes that multipole components have already been rotated into the global coordinate frame

**INDUCE0A Subroutine**

"induce0a" computes the induced dipole moment at each polarizable site using a pairwise double loop

**INDUCE0B Subroutine**

"induce0b" computes the induced dipole moment at each polarizable site using a regular Ewald summation

**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

**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

**INITPRM Subroutine**

"initprm" completely initializes a force field by setting all parameters to zero and using defaults for control values

**INITRES Subroutine**

"initres" sets names for biopolymer residue types 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; assumes internal coordinates have already been setup

**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

**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

**ISPLPE Subroutine**

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

**JACOBI Subroutine**

"jacobi" performs a matrix diagonalization of a real symmetric matrix by the method of Jacobi rotations

**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

**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

**KCHARGE Subroutine**

"kcharge" assigns partial charges to the atoms within the structure and processes any new or changed values

**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

**KENEG Subroutine**

"keneg" applies primary and secondary electronegativity bond length corrections to applicable bond parameters

**KEWALD Subroutine**

"kewald" assigns both regular Ewald summation and particle mesh Ewald parameters for a periodic box

**KGEOM Subroutine**

"kgeom" asisgns parameters for geometric restraint terms to be included in the potential energy calculation

**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

**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

**KOPBEND Subroutine**

"kopbend" assigns the force constants for out-of-plane bending at trigonal centers via Wilson-Decius-Cross angle bends; also processes any new or changed parameter values

**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

**KPITORS Subroutine**

"kpitors" assigns pi-orbital 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

**KSOLV Subroutine**

"ksolv" assigns continuum solvation energy parameters for the Eisenberg-McLachlan ASP, Ooi-Scheraga SASA or various GB/SA solvation models

**KSTRBND Subroutine**

"kstrbnd" assigns the parameters for the stretch-bend interactions and processes new or changed parameter values

**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

**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 the Levenberg-Marquardt step during a nonlinear least squares calculation; this version is based upon ideas from the Minpack routine LMPAR together with with 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

**MAKEINT Subroutine**

"makeint" converts Cartesian to internal coordinates where selection of internal coordinates is controlled by "mode"

**MAKEPDB Subroutine**

"makexyz" converts a set of Cartesian coordinates to Protein Data Bank format with special handling for systems consisting of polypeptide 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

**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

**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 and induced dipole information; 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**

**MEASFP 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

**MMID Subroutine**

"mmid" implements a modified midpoint method to advance the integration of a set of first order differential equations

**MODECART Subroutine**

**MODEROT Subroutine**

**MODESRCH Subroutine**

**MODETORS Subroutine**

**MODULI Subroutine**

"moduli" sets the moduli of the inverse discrete Fourier transform of the B-splines; bsmod[1-3] hold these values, nfft[1-3] are the grid dimensions, bsorder is the order of B-spline approximation

**MOLECULE Subroutine**

"molecule" counts the molecules, assigns each atom to its molecule and computes the mass of each molecule

**MOLUIND Subroutine**

"moluind" computes the molecular induced dipole components in the presence of an external electric field

**MOMENTS Subroutine**

"moments" computes the total electric charge, dipole and quadrupole moments for the entire system as a sum over the partial charges, bond dipoles and atomic multipole moments

**MONTE Program**

"monte" performs a Monte Carlo/MCM conformational search using either 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"

**NEEDUPDATE Subroutine**

**NEIGHBOR Subroutine**

"neighbor" finds all of the neighbors of each atom

**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

**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 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

**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

**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

**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 two 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

**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 "t"

**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

**PDBATM Subroutine**

"pdbatm" 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" first modifies bond lengths and force constants according to the standard bond slope parameters and the bond order values stored in "pnpl"; also alters some 2-fold torsional parameters based on the bond-order \* beta matrix

**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 the pisystem using a modified Pariser-Parr-Pople method

**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

**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

**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

**POTNRG Function**

**POTOFF Subroutine**

"potoff" clears the forcefield definition by turning off the use of each of the potential energy functions

**POWER Subroutine**

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

**PRECISE Function**

"precise" finds a machine precision value as selected by the input argument: (1) the smallest positive floating point value, (2) the smallest relative floating point spacing, (3) the largest relative floating point spacing

**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 by scaling the coordinates via coupling to an external constant pressure bath

**PRMKEY Subroutine**

"field" parses a text string to extract keywords related to force field potential energy functional forms and constants

**PROCHAIN Subroutine**

"prochain" builds up the internal coordinates for an amino acid sequence from the phi, psi, omega and chi values

**PROJCT Subroutine**

**PROMO Subroutine**

"promo" writes a short message containing information about the TINKER version number and the 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

**PROPYZE Subroutine**

"propyze" finds and prints the total charge, dipole moment components, radius of gyration and moments of inertia

**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

**PRTCAR Subroutine**

"prtcar" writes out a set of Cartesian coordinates for all active atoms in the Accelerys InsightII .car 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

**PRTINT Subroutine**

"prtint" writes out a set of Z-matrix internal coordinates to an external disk file

**PRTMOL2 Program**

"prtmol2" writes out a set of coordinates in Sybyl MOL2 format to an external disk file

**PRTPDB Subroutine**

"prtpdb" writes out a set of Protein Data Bank coordinates to an external disk file

**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

**PRTXMOL Subroutine**

"prtxmol" writes out a set of Cartesian coordinates for all active atoms in a simple, generic XYZ format originally used by the XMOL program

**PRTXYZ Subroutine**

"prtxyz" writes out a set of Cartesian coordinates to an external disk file

**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**

**PTINCY Function**

**PZEXTR Subroutine**

"pzextr" is a polynomial extrapolation routine used during Bulirsch-Stoer integration of ordinary differential equations

**QRFACT Subroutine**

"qrfact" performs Householder transformations with column pivoting (optional) to compute a QR factorization of the m by n matrix a; 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; normally used in combination 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

**READINT Subroutine**

"readint" gets a set of Z-matrix internal coordinates from an external file

**READMOL2 Subroutine**

"readmol2" gets a set of Sybyl MOL2 coordinates from an external disk file

**READPDB Subroutine**

"readpdb" gets a set of Protein Data Bank coordinates 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

**RFINDEX Subroutine**

"rfindex" finds indices for each multipole site for use in computing reaction field energetics

**RGDSRCH Subroutine**

**RGDSTEP Subroutine**

"rgdstep" performs a single molecular dynamics time step for a rigid body calculation

**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

**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

**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 converts from the local coordinate system to 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" computes the atomic multipoles at a specified site in 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

**SCAN Program**

"scan" attempts to find all the local minima on a potential energy surface via an iterative series of local searches

**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

**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 a velocity Verlet integration algorithm

**SDTERM Subroutine**

"sdterm" gets frictional and random force terms needed to update positions and velocities via stochastic dynamics

**SEARCH Subroutine**

"search" is a unidimensional line search based upon parabolic extrapolation and cubic interpolation using both function and gradient values; if forced to search in an uphill direction, return is after the initial step

**SETACCELERATION Subroutine**

**SETATOMIC Subroutine**

**SETATOMTYPES Subroutine**

**SETCHARGE Subroutine**

**SETCONNECTIVITY Subroutine**

**SETCOORDINATES Subroutine**

**SETENERGY Subroutine**

**SETFILE Subroutine**

**SETFORCEFIELD Subroutine**

**SETGRADIENTS Subroutine**

**SETIME Subroutine**

"setime" initializes the elapsed interval CPU timer

**SETINDUCED Subroutine**

**SETKEYWORD Subroutine**

**SETMASS Subroutine**

**SETNAME Subroutine**

**SETSTEP Subroutine**

**SETSTORY Subroutine**

**SETTIME Subroutine**

**SETUPDATED Subroutine**

**SETVELOCITY Subroutine**

**SHAKEUP Subroutine**

"shakeup" initializes any holonomic constraints for use with the rattle algorithm during molecular dynamics

**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

**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

**SMOOTH Subroutine**

"smooth" sets the type of smoothing method and the extent of surface deformation for use with potential energy smoothing

**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

**SQUARE Subroutine**

"square" is a nonlinear least squares routine derived from the IMSL routine BCLSF and More's Minpack routine LMDER; 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 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

**SURFATOM Subroutine**

"surfatom" performs an analytical computation of the surface area of a specified atom; a simplified version of "surface"

**SWITCH Subroutine**

"switch" sets the coeffcients used by the fifth and seventh order polynomial switching functions for spherical cutoffs

**SYBYLXYZ Program**

"sybylxyz" takes as input a Sybyl MOL2 coordinates file, converts to and then writes out Cartesian coordinates

**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

**TANGENT Subroutine**

"tangent" finds the projected gradient on the synchronous transit path for a point along the transit pathway

**TEMPER Subroutine**

"temper" applies a velocity correction at the half time step as needed for the Nose-Hoover extended system thermostat

**TEMPER2 Subroutine**

"temper2" computes the instantaneous temperature and applies a thermostat via Berendsen velocity scaling, Andersen stochastic collisions, Langevin piston or Nose-Hoover extended systems

**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

**TESTLIGHT Program**

"testlight" performs a set of timing tests to compare the evaluation of potential energy and energy/gradient using the method of lights with a double loop over all atom pairs

**TESTROT Program**

"testrot" computes and compares the analytical and numerical gradient vectors of the potential energy function with respect to rotatable torsional angles

**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)

**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 sites defined by local coordinate frames and distributes thme to convert to forces on the original sites and sites specifying the local frames

**TORQUE1 Subroutine**

"torque1" takes the torque value on a site defined by a local coordinate frame and distributes it to convert to forces on the original site and sites specifying the local frame

**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

**TORSIONS Subroutine**

"torsions" finds the total number of dihedral angles and the numbers of the four atoms defining each dihedral 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

**TRANSIT Function**

"transit" evaluates the synchronous transit function and gradient; linear and quadratic transit paths are available

**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

**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; this version is based on the ideas found in NL2SOL and in Dennis and Schnabel's book

**UDIRECT1 Subroutine**

"udirect1" computes the reciprocal space contribution of the permanent atomic multipole moments to the electrostatic field for use in finding the direct induced dipole moments via a regular Ewald summation

**UDIRECT2 Subroutine**

"udirect2" computes the real space contribution of the permanent atomic multipole moments to the electrostatic field for use in finding the direct induced dipole moments via a regular Ewald summation

**UFIELD Subroutine**

"ufield" finds the field at each polarizable site due to the induced dipoles at the other sites using Thole's method to damp the field at close range

**UMUTUAL1 Subroutine**

"umutual1" computes the reciprocal space contribution of the induced atomic dipole moments to the electrostatic field for use in iterative calculation of induced dipole moments via a regular Ewald summation

**UMUTUAL2 Subroutine**

"umutual2" computes the real space contribution of the induced atomic dipole moments to the electrostatic field for use in iterative calculation of induced dipole moments via a regular Ewald summation

**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

**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

**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

**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 by means of 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

**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

**VIBRIGID Program**

"vibrigid" computes the eigenvalues and eigenvectors of the Hessian matrix over rigid body degrees of freedom

**VIBROT Program**

"vibrot" computes the eigenvalues and eigenvectors of the torsional Hessian matrix

**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

**XTALERR Subroutine**

"xtalerr" computes an error function value derived from derivatives with respect to lattice parameters, lattice energy and monomer dipole moments

**XTALFIT Program**

"xtalfit" computes an optimized set of potential energy parameters for user specified van der Waals and electrostatic interactions by fitting to crystal structure, lattice energy and monomer dipole moment data

**XTALLAT1 Function**

"xtalmol1" is a service routine that computes the energy and numerical gradient with respect to the six lattice lengths and angles for a crystal energy minimization

**XTALMIN Program**

"xtalmin" performs a full crystal energy minimization by alternating cycles of truncated Newton optimization over atomic coordinates with variable metric optimization over the six lattice dimensions and angles

**XTALMOL1 Function**

"xtalmol1" is a service routine that computes the energy and gradient with respect to the atomic Cartesian coordinates for a crystal energy minimization

**XTALMOL2 Subroutine**

"xtalmol2" is a service routine that computes the sparse matrix Hessian elements with respect to the atomic Cartesian coordinates for a crystal energy minimization

**XTALMOVE Subroutine**

"xtalmove" converts fractional to Cartesian coordinates for rigid molecules during fitting of force field parameters to crystal structure data

**XTALPRM Subroutine**

"xtalprm" stores or retrieves a crystal structure; used to make a previously stored structure the currently active structure, or to store a structure for later use; only provides for the intermolecular energy terms

**XTALWRT Subroutine**

"xtalwrt" is a utility that prints intermediate results during fitting of force field parameters to crystal data

**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 a Cartesian coordinates file

**XYZINT Program**

"xyzint" takes as input a Cartesian coordinates file, then converts to and writes out an internal coordinates 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

**XYZSYBYL Program**

"xyzsybyl" takes as input a Cartesian coordinates file, converts to and then writes out a Sybyl MOL2 file

**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