Tabulated MC for Proteins – Internals

Justin Spiriti

December 1, 2016

1 General

Double precision arithmetic is used in the program throughout, except for storing table entries (where single precision saves memory) or when writing DCD files (where it is part of the file format).

Cartesian coordinates are stored as a one-dimensional array in the order $x_1, y_1, z_1, x_2, y_2, z_2, \ldots$. Rotations are represented by normalized quaternions with positive real part, and are stored with the real part first, followed by the x, y, and z components.

Most of the basic math routines are found in `rotations.cpp`. These include converting back and forth between quaternions, matrices, Euler angles, and axis-angle representations, performing quaternion and matrix arithmetic, and performing RMSD fitting, among many other things. Note that if we have two rotations 1 and 2, the quaternion representing performing rotation 1 followed by 2 is given by the forward product $\mathbf{q}_1\mathbf{q}_2$, but if using rotation matrices the corresponding matrix is given by the reverse product R_2R_1 (that is to say, quaternions are multiplied in the same order as the rotations are performed, while matrices are multiplied in the reverse order).

The system is divided into fragments, each of which is one of several fragment types. Each fragment type is identified by a name and a numerical index which points to a `fragmenttype` object in the `top->fragtypes` array. The `tables` array is a two dimensional array of pointers to table objects, which is indexed by pairs of type numbers.

Throughout most of the code, individual atoms are identified by their index in the overall system, from 0 to `top->natom-1`. However, each atom also has an index corresponding to its position within the fragment. In some cases it is necessary to rearrange coordinates of atoms from the ordering they have in the whole system to the ordering they have in an individual fragment.

The `io.cpp` file contains subroutines for performing I/O operations. Many of the constructors for objects also take file names and read information from those files.

2 class `simulation`

This class represents a simulation, and is defined in `mc.h`. Besides information on the current configuration, there are many members giving simulation parameters, and open file handles to the trajectory output files.

2.1 Major fields

Name	Description
`oldcenter`, `oldorient`, `oldcoords`	The current fragment centers, orientations and atomic coordinates, before the current MC move.
`newcenter`, `neworient`, `newcoords`	The current centers, orientations and atomic coordinates, after the current MC move.)
`top`	Topology object (of class `topology`).
`ffield`	Force field object (of class `forcefield`).
`tables`	Two-dimensional array of table objects, indexed by pairs of fragment type numbers.

2.2 Major methods

Name	File	Description
`simulation::mcloop`	`mc.cpp`	Main Monte Carlo loop
`simulation::mcmove`	`mcmoves.cpp`	Performs MC moves (master routine, calls several others)
`simulation::total_energy`	`mc.cpp`	Calculates total energy
`simulation::moved_energy`	`mc.cpp`	Calculates the part of energy needed for MC moves (not currently used)
`simulation::interaction_energy`	`mc.cpp`	Determinse whether to calculate interaction energy exactly or using tables, which table is needed, and which fragment will be the reference fragment
`simulation::simulation`	`init.cpp`	Supervises most of the initialization, including processing input for simulations, reading in all files, and fitting fragments to the initial structure
`simulation::calculate_born_radii`	`solvation.cpp`	Supervise the calculation of Born radii. (GB version only.)

3 class `topology`

This class represents topology information for the current system and is defined in `topology.h`.

3.1 Major fields

Name	Description
`natom`	Total number of atoms in the system.
`atoms`	Array of structures giving information on individual atoms. The `fragment` member indicates the fragment to which a given atom belongs. The `fragatom` member indicates the atom's index within its fragment.
`nfrag`	Total number of fragments in the system.
`frags`	Array of structures giving information on fragments. The `type` member gives the fragment type number, while the `main_chain_prev`, `main_chain_next`, `side_chain_prev`, and `side_chain_next` members can be used to walk the protein backbone or side chain. The `atoms` member is an array giving the system atom numbers for all atoms within the fragment, indexed by their index within the fragment.
`nfragtypes`	Total number of fragment types available.
`fragtypes`	Array of fragment type objects, indexed by type number.

3.2 Major methods

Name	File	Description
`topology::frag_type_by_name`	`topology.cpp`	Find the index number corresponding to a fragment type name.
`topology::assemble_fragments`	`topology.cpp`	Supervises RMSD fitting of each fragment in its initial structure to its reference geometry.
`topology::update_coords`	`topology.cpp`	Updates atomic coordinates for one fragment, given the fragment center and orientation.
`topology::exact_interaction_energy`	`mc.cpp`	Wrapper for `forcefield::exact_interaction_energy`, rearranging coordinates so that they correspond to the atoms within a fragment.

4 class `fragmenttype`

This class represents information on an individual fragment type and is defined in `fragments.h`. Many methods of this class expect atoms in "fragment order" rather than "system order", so it may be necessary to perform a reordering.

4.1 Major fields

Name	Description
`fragname`	Fragment type name.
`names`	Individual atom names within fragment.
`types`	Individual atom type numbers.
`refgeom`	Cartesian coordinates of the reference geometry.

4.2 Major methods

Name	File	Description
`fragmenttype::get_coords`	`fragments.cpp`	Calculate Cartesian coordinates of atoms from the center and orientation of a fragment.
`fragmenttype::fit_fragment`	`fragments.cpp`	Determine position and orientation of one fragment by RMSD fitting to atomic coordinates.
`fragmenttype::get_average_born_radius`	`solvation.cpp`	Compute the "average" born radius for a fragment. (GB version only.)

5 class `table`

This class represents a table and is defined in `tables.h`.

5.1 Major fields

Name	Description
`hdr`	A structure (of type `table_header`) that comprises the header at the beginning of each binary table file and contains information on the resolution of the table, the force field and parameters used to generate it, and the fragments involved.
`energy`	An array of `floats` that contains the actual energy values in the table

5.2 Major methods

Name	File	Description
`table::table_interaction_energy`	`tables.cpp`	Supervises computation of net displacement and relative orientation and table lookup
`table::get_energy`	`tables.cpp`	Determines table indices $(n_r, n_\theta, n_\phi, n_{\phi'}, n_{\theta'}, n_{\psi'})$
`table::calculate_index`	`tables.h`	Calculates overall table index from table indices $(n_r, n_\theta, n_\phi, n_{\phi'}, n_{\theta'}, n_{\psi'})$
`table::read_table_header_info`	`tablegen.cpp`	Reads and processes table header information from input file, defines grid for table.
`table::print_header_info`	`tables.cpp`	Prints information from the table header
`table::fill_table`	`tablegen.cpp`	Fills the table with interaction energies
`table::boltzmann_average_trans`	`tablegen.cpp`	Performs translational smoothing
`table::boltzmann_average_orient`	`tablegen.cpp`	Performs orientational smoothing
`table::generate_table`	`tablegen.cpp`	Supervises table generation.
`table::alloc_read_table`	`tables.cpp`	Reads a table from a binary file. (Constructs a memory mapping for the table file if the symbol `NO_MMAP_TABLES` is not defined.)
`table::write_table`	`tables.cpp`	Writes a binary file containing a table.

6 class `forcefield`

This class contains information about the forcefield and is defined in `ffield.h`.

6.1 Major fields

Name	Description
`vdwParams`	Van der Waals parameters for each atom type.
`bondParams`	Parameters for the bonding term of the force field.
`angleParams`	Parameters for the angle term of the force field.
`dihedParams`	Parameters for the dihedral term of the force field.
`impropParams`	Parameters for the improper dihedral term of the force field.

6.2 Major methods

`forcefield::exact_interaction_energy`	`ffield.cpp`	Calculates the interaction
`forcefield::non_tab_energy`	`ffield.cpp`	Calculates all non-tabulated parts of the energy
`forcefield::moved_non_tab_energy`	`ffield.cpp`	Calculates parts of non-tabulated energy (not currently used)