Browndye 2.0 User's Manual

Gary Huber

October 6, 2023

1 Requirements

This package requires a C++ compiler that can compile the C++17 standard; this includes the current g++ and clang++. It also requires an Ocaml compiler. External libraries required are Expat for XML parsing, and Lapack for linear algebra; often these are already built into Linux distributions, but are easy to install if not. Depending on the state of the C++ library, the Boost C++ library may be required; this currently appears to be the case for the Mac. By default, the GNU g++ library is used and the Boost library is not required, but look at the top of the Makefile in the main directory for instructions on how to change it.

2 Functionality

The Browndye software package consists of two simulation programs and several auxiliary programs for processing data. It can perform Brownian dynamics (BD) simulations on collections of of *cores*, large rigid bodies composed of many smaller spheres, and *chains*, flexible assemblies of spheres that can interact with each other and the cores with user-specified force fields. Several cores and chains can be assembled into *groups*. It has functionality very similar to the packages SDA, GeomBD2, and MacroDox. In addition to outputting trajectories, it can also compute the 2nd-order rate constant of the encounter of two groups, and relative probabilities of transitions from one conformation or binding mode to another.

2.1 Cores

Each core is made up of charged spheres organized as *atoms* grouped into *residues*. The atoms don't necessarily have to represent actual atoms; they could, for example, represent interaction sites in coarse-grained models. Each core use one or more electric potential grids from APBS (www.poissonboltzman.org) for computing electrostatic forces with other cores and chains. Each atom can interact with atoms on other cores and chains by means of a Lennard-Jones potential or a potential defined by tabulated data and evaluated by a cubic spline. Each core can also have one or more grids of precomputed Born $1/r^4$ integral values for approximately computing polar desolvation forces with other cores.

The atoms in each core are organized into recursive nested spheres in order to allow quick computation of short-ranged forces, without having to loop over all pairs of atoms. The charges on the atoms are lumped together when appropriate in order to save time in computing Coulombic interactions.

2.2 Chains

Each chain is made of spherical *atoms* joined together with bonds. In addition to two-atom bonds, three-atom bond angles and four-atom dihedral angles can also be defined. Forces involving the bonds, bond angles, and dihedral angles can be computed using either a force field using a conventional form from molecular mechanics, or from a cubic spline using tabulated data. Likewise, the pair-wise non-bonded forces can follow a Lennard-Jones form or a tabulated spline as described above for the cores. In addition, rigid distance constraints between two atoms and coplanar constraints among four atoms can be imposed. Each chain can also be joined to atoms on one or more cores through bonds, bond angles, dihedral angles, or constraints. Each chain can also be "frozen" and treated as a rigid body when it is far from a specified site on another core or chain, in order to allow larger time steps. Furthermore, the simulation code will freeze an entire connected collection of chains and cores and treat it as a single rigid body, if all the the chains are frozen.

2.3 Dummies

In order to define reactions (explained below), it is sometimes useful to define dummy atoms, which have no physical interactions but serve as reference points. They are grouped into a dummy "molecule", which is tied to a particular core. For example, if Molecule B binds to Molecule A, a dummy molecule might consist of a copy of a subset of Molecule B atoms that touch Molecule A in the binding pose, and it would move rigidly with a core of Molecule A. A reaction would then be defined by distances between each dummy atom and its corresponding atom on Molecule B.

2.4 Motion

The rigid cores and flexible chains are stepped forward using the equations of Brownian dynamics after the forces are computed [3]. When rigid constraints are imposed, the forces necessary to maintain the constraints are automatically computed at each time step. The time step size is adaptive, with reasonable guesses made based on the potential energy functions and geometry, and corrections are made if the forces change too quickly [7]. Approximate hydrodynamics among chains and cores are computed following the method of Elcock [2]; each core is treated as a tetrahedron of four spheres that gives the desired translational and rotational diffusivities.

2.5 Forces

All electrostatic forces involving the cores are computed from APBS electric potential grids computed for each core. Given a set of grids on a core, the other cores interact with it via a set of point charges, which can be adaptively lumped during the simulation [7]. The point charges for each core are read into the simulation programs separately from the residues and atoms, because certain models call for lumping or adjusting of charges, and it is easier to do that outside of the simulation programs. However, the atoms in a chain have point charges at their centers and are read in directly from the atom files. The point charges in a chain interact pairwise with other chain atoms through the screened Coulombic potential,

$$V(r) = \frac{\exp(-\frac{r}{\lambda_d})}{4\pi\alpha_s\epsilon_0 r} \tag{1}$$

where λ_d and α_s are the Debye length and relative dielectric constant of the solvent. Finally, the chains interact with each core through its electric potential grid.

Short-ranged non-bonded interactions are described either through a Lennard-Jones potential

$$V(r) = \frac{A}{r^{12}} - \frac{B}{r^6} \tag{2}$$

alternately described as

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{3}$$

or a tabulated cubic spline in the manner of the COFFDROP potential [1, 4]. These interactions are applied pairwise among the atoms of each core and chain, and the atoms on each core are recursively grouped into larger spherical domains in order to avoid unnecessary looping over core atoms that are out of range of other atoms [7].

Each atom and residue has a *type*, and each pair of atom and residue types is converted into an internal integer atom type. Pairs of these integer atom types are used to select the particular pair potential for each atom pair. For the tabulated potentials, all potentials for each pair of (atom type, residue type) must be included, although wildcard values for the residue types can be included in the parameter file, and redundant tables are not stored separately.

The Lennard-Jones parameters can be specified in three different ways. If two atoms interact with each other, and the Lennard-Jones A and B parameters have been specified for that particular pair of internal atom types (where the internal atom type is specified by both residue type and atom type within the residue), then that potential is used. If the Lennard-Jones parameters have been specified for each atom type separately but not together, then the Lorenz-Bertholet mixing rules are applied to generate parameters for the pair. Finally, if an atom's type is not listed in the parameter file, then a purely repulsive interaction is assumed with $B = 0$, and an approximation to the hard-sphere interaction is made using the atom's radius. The A parameter is chosen from the atom radius so that the "hole" left by the atom in a cloud of points interacting with it by the r^{-12} potential in thermal equilibrium at 298 K would have the same radial second moment as a hard sphere of the given radius.

$$A = k_b T (0.9345 R)^{12} \tag{4}$$

(The explicit hard-sphere interactions from the previous versions of Browndye are no longer used.)

The bonded interactions are specified by pairs of atoms for bonds, trios of atoms for bond angles, and quartets of atoms for dihedral angles. Core atoms can also be included in these structures, but at least one atom must be a chain atoms. The tabulated functions are specified by evenly-spaced values of distance for the bonds, or angle for the bond and dihedral angles. Each bond, bond angle, and dihedral angle has a type that is specified in the parameter file. The molecular mechanics form for bonded interactions is

$$\begin{aligned} V = \frac{1}{2}\Big[&\sum_i k_{b,i}(r_i - r_{eq,i})^2 \\ &+ \sum_j k_{\theta,j}(\theta_j - \theta_{eq,j})^2 \\ &+ \sum_k \sum_m V_{k,m}(1 + \cos(n_{k,m}\phi_k - \gamma_{k,m}))\Big] \end{aligned} \tag{5}$$

where the three terms on the right side are sums over bonds, bond angles, and dihedral angles, respectively, and the parameters come from the particular type of bonded structure.

Unfortunately, at this time it is not possible to combine the tabulated spline forces with the molecular mechanics-style forces (Lennard-Jones and Eq. 5) in the same simulation; the user must choose one or the other.

The length constraints can be specified by a pair of atoms and a distance and the constraint requiring four atoms to lie in a plane is specified by listing the four atoms. A constraint may include core atoms, but must have at least one chain atom. The constraints can be used with either set of force models.

The polar desolvation forces between two cores are computed as in previous versions of Browndye, with squared charges of one core interacting with the following integral on another core:

$$\frac{3(\alpha_s - \alpha_p)}{4\pi(2\alpha_s + \alpha_p)\alpha_s\epsilon_0} \int (1 + \kappa r)^2 \exp(-2\kappa r) r^{-4} dr \tag{6}$$

where the integral is over all interior points of one core, r is the distance from the charge on the other core to the interior point, and α_p and α_s are the relative permitivities of the core and the solvent. The inverse Debye length is κ. The integral for each point is computed ahead of time and stored on a grid. This has the same form as the formula used in SDA, where the integral is approximated by a sum over spheres. The user can also input a factor by which to multiply this force.

Also included, as an option, are nonpolar desolvation, or hydrophobic terms, as approximated in Gabdoulline and Wade (2009) by a pair-wise term in Eqs. 1 and 3 of the reference.

$$\Delta G_{npd,ij} = A_i \Phi_j(r_{ij}) \tag{7}$$

where A_i is a precalculated estimate of the solvent-accessible area of Atom i, and

$$\Phi(r_{ij}) = \beta c \begin{cases} 1, & r < a \\ \frac{b-r}{b-a}, & a < r < b \\ 0, & r > b \end{cases} \tag{8}$$

is an estimate of how buried Atom j is by Atom i, times a surface tension factor β. The distance r_{ij} is the distance from the VdW surface of Atom j to the center of Atom i, and the parameters have the default values $a = 3.1\AA$, $b = 4.35\AA$, $c = 0.5$, and $\beta = 0.025\ kcal\ mol^{-1}\AA^{-2}$ (which is converted to Browndye units). These values can be changed (see below), and use of this energy term is triggered by including the precalculated SASA values (see below). This term is used only with the molecular mechanics form of the energy, but not with the spline-based energy.

Browndye also has the option of using an external density, perhaps from microscopy or tomography, to guide the positions of the atoms. This method can be used to dock x-ray crystallography structures into a lower-resolution map. [10] This is described briefly in the **density_field** entry below.

2.6 Reactions

The second-order association rate constant can be computed for two groups by using the Northrup-Allison-McCammon (NAM) algorithm [8]. The simulation code selects the b-radius of the algorithm by finding the distance at which the forces become sufficiently spherically symmetric. The very first conformations of the groups are used and the b-radius is then fixed, so if a group changes shape significantly during a simulation, it is possible that the b-radius might no longer be valid for computing a rate constant. The user can also input a b-radius. The outer q-radius is also chosen by the simulation code, and a great enough distance is chosen so that the chance of the two groups returning to within the b-radius is negligible. This is not a problem, since the code uses a simplified model with adaptive time steps to resolve the outer regions, thereby using negligible simulation time. The weighted-ensemble method can be used with this geometry [6], with copies started on the b-surface and bins spanning the space between the b-surface and the reaction.

The probability of one or two groups moving from one conformation to another can also be computed. If two groups are used, the NAM algorithm can be used to compute the probability of escape before reaching another conformation.

A reaction criterion is specified by pairs of atoms. Each pair is also assigned a center-to-center distance. The reaction itself is assigned a number n. A pair is said to be "satisfied" if its atoms are closer than the assigned distance. If n pairs are satisfied, then the reaction is said to occur. For example, a criterion might comprise all pairs of atoms forming hydrogen bonds in a protein-protein complex, and the requirement that any three of the pairs are within 5 $\mathring{A}$. In addition to containing pairs, a reaction criterion can contain other reaction criteria; these inner criteria are treated like pairs to be satisfied. An example would be an enzyme with two active sites. The binding of the substrate to each site would be represented by a separate reaction criterion, but the two reaction criteria could be lumped into another criterion to represent the overall reaction.

For a reaction criterion with only atom pairs, the *reaction coordinate* is defined to be the n^{th} smallest difference, among the pairs, between the actual interatomic distance and the required distance for that pair. So, at reaction, the reaction coordinate is zero. For a criterion containing other criteria, the same formula applies, except the reaction coordinates of the contained criteria are treated like the distance differences of the pairs.

Entire reaction pathways can be specified by named *states* and named *reactions*. You can think of states as being points connected by arrows, which are the reactions. The simplest example would be the bimolecular reaction, in which the system passed from a "reactants" state to a "products" state by passing through the one reaction. An escape can also occur, but that does not count as a reaction.

2.7 Units

Browndye uses physical units of length in Ångstroms, time in picoseconds, energy in multiples of $k_B T$ where k_B is Bolzmann's constant and the temperature T is $298K$, and charge in multiples of the elementary charge. These are the same units used by APBS. However, the 2nd-order rate constants are output in units of 1/(M s).

2.8 Other

Browndye can run on several CPU cores in shared-memory mode, which trajectories distributed across the cores, both when using single trajectories and using the weighted-ensemble method. At this time, the weighted ensemble method can only be used for a single association reaction but not for more complex reaction pathways. There are two main simulation programs: **nam_simulation**, which runs one trajectory at a time, and **we_simulation**, which uses the weighted-ensemble method. There are also several auxilliary programs which can be used to prepare the input, compute rate constants from the output, and prepare trajectory files for viewing with VMD. The starting information required for a Browndye simulation are electrostatic grids from APBS, PQR files of the atoms of the rigid cores and chains, files describing the chains and how they are connected to the cores, files describing the reaction criteria, and a file describing the simulation itself. All of the the files use the XML format, except for the large grid files, which use the DX format. The only external libraries required for compiling Browndye are the Expat XML parser, the C interface Lapacke to Lapack, and the C++ Boost libraries.

3 Simulation Program Usage

This section will go through the main input file, and the various files that it refers to. Later on we will discuss the auxilliary programs that are used to create some of the files.

3.1 Installation Requirements

The following freely available pieces of software are required for Browndye:

- C++ compiler that is up-to-date enough to handle C++14 code
- compiler for the OCaml programming language
- Expat XML Parser
- C++ Boost library
- LAPACK linear algebra library
- LAPACKE C interface to LAPACK

By default, the compilation method assumes that the C++ compiler used is g++, the GNU compiler.

3.2 Simulation File

The first is the *simulation file*, which is given directly to the simulation code.

```
<root>
  <n_threads> integer </n_threads> # optional
  <seed> integer </seed>
  <output> filename </output>

  # for nam_simulation
  <n_trajectories> integer </n_trajectories>
  <n_trajectories_per_output> integer </n_trajectories_per_output> # op
  <max_n_steps> integer </max_n_steps>
  <trajectory_file> filename </trajectory_file> # optional
  <n_steps_per_output> integer </n_steps_per_output> # optional
  <min_rxn_dist_file> filename </min_rxn_dist_file> # optional

  # for we_simulation
  <n_copies> integer </n_copies>
  <n_bin_copies> integer </n_bin_copies>
  <n_steps> integer </n_steps>
  <n_we_steps_per_output> integer </n_we_steps_per_output>
  <bin_file> filename </bin_file>

  # system description
```

```
<system>
  <solvent_file> filename </solvent_file>
  <force_field> name </force_field>
  <parameters> filename </parameters>
  <start_at_site> true/false </start_at_site> # optional
  <b_radius> real </b_radius> # optional
  <reaction_file> filename </reaction_file> # optional
  <no_nam> true/false </no_nam> # optional

  <density_field> filename </density_field> # optional
  <atom_weights> filename </atom_weights> # optional

  <hydrodynamic_interactions> true/false </hydrodynamic_interactions>
  <n_steps_between_hi_updates> integer <n_steps_between_hi_updates> #

  <time_step_tolerances> # optional
    <force> real </force>
    <reaction> real </reaction>
    <minimum_core_dt> real </minimum_core_dt>
    <minimum_chain_dt> real </minimum_chain_dt>
    <minimum_core_reaction_dt> real </minimum_core_reaction_dt>
    <minimum_chain_reaction_dt> real </minimum_chain_reaction_dt>
    <constant_dt> real </constant_dt> # optional
  </time_step_tolerances>

  <restraints> # optional
    <restraint> # one or more
      <length> real </length>
      <group0> name </group0>
      <core0> name </core0>
      <chain0> name </chain0> # either core0 or chain0, but not both
      <atom0> integer </atom0>
      <group1> name </group1>
      <core1> name </core1>
      <chain1> name </chain1> # either core1 or chain1, but not both
      <atom1> integer </atom1>
    </restraint>
  </restraints>
```

```
<group> # one or more
  <name> name </name>
  <core> # zero or more
    <name> name </name>
    <atoms> filename </atoms>
    <hydro_params> filename </hydro_params>

    <electric_field> # optional
      <grid> filename </grid> # one or more
      <multipole_field> filename </multipole_field>
    </electric_field>

    <desolvation_field> # optional
      <grid> filename </grid> # one or more
    </desolvation_field>

    <eff_charges> filename </eff_charges> # optional
    <eff_charges_squared> filename </eff_charges_squared> # optiona

    <areas> filename </areas> # optional

    <copy>
      <core> core name </core>
      <translation> 3 reals </translation>
      <rotation> 9 reals </rotation>
    </copy>

  </core>

  <chain> filename </chain> # zero or more

  <dummy> # zero or more
    <name> name </name>
    <atoms> filename </atoms>
    <core> name </core>
  </dummy
```

```
    </group>

  </system>

</root>
```

The entries are explained as follows:

- **n_threads** - number of threads used; should not be greater than the number of available CPU cores. Default is 1.
- **seed** - used to seed the random number generator
- **output** - name of file where output is sent (not trajectories)
- The following are used for **nam_simulation**.
- **n_trajectories** - total number of trajectories to be run
- **n_trajectories_per_output** - number of time steps between updates to the output file. Default is 1.
- **max_n_steps** - number of time steps taken in a trajectory before giving up and declaring that it is "stuck".
- **trajectory_file** - prefix of files used for outputting of trajectories. A different file is created for each thread. If the prefix is "trajectory", then the file for Thread 0 is "trajectory0.xml", and so forth. This is optional; if it is not given, then no files are generated. The trajectory file consists of rotation matrices and translation vectors describing the state of the rigid cores, and positions of the atoms in the chains.
- **n_steps_per_output** - number of time steps between output to trajectory file. Default is 1.
- **n_output_states_per_block** - System states are lumped together in a MIME-formatted data block upon output to a trajectory file, in order to save space in the XML file. This integer tells how many system states are to be included in a block. The larger the number, the more efficient the output might be, but the more memory is required to hold the state data before it is output. Default is 1000.

- **min_rxn_dist_file** - file to output minimum reaction coordinate for each trajectory.
- The following are used for **we_simulation**.
- **n_bin_copies** - number of system copies used in building bins
- **bin_file** - file containing the bin information, comes from the program **build_bins**
- **n_copies** - number of system copies
- **n_steps** - total number of steps; if you have m copies and you do n time steps, the total number of steps is n*m
- **n_we_steps_per_output** - number of time steps between updates of output file
- **system** information about the physical system and its motions are described inside this section.
    - **solvent_file** - information on the solvent
    - **force_field** - for now, can take the values **molecular_mechanics**, **molecular_mechanics_hp** or **spline**, to select the type of force field. The option **molecular_mechanics_hp** includes the hydrophobic force option above.
    - **parameters** - parameter file for the force field. Optional for **molecular_mechanics** with no chains.
    - **start_at_site** - **true** or **false**; if true, then then all the molecular components are started according to their atom input files; otherwise, the second group is started on the b-sphere and the second-order rate constant is computed.
    - **b_radius** - This can be automatically computed, but if a value is given, it overrides the automatic value.
    - **hydrodynamic_interactions** - **true** or **false**; include hydrodynamic interactions. Default is **true**.
    - **n_steps_between_hi_updates** - number of steps between updates to hydrodynamic interactions. For large chains, this can take most of the simulation time. Default is one.

- **reaction_file** - describes the reaction pathways and criteria
- **no_nam** - If and only if you have two groups and a reaction, the simulation programs will automatically construct a b-radius and assume the possibility of escape. However, if you have two groups restrained to each other along with a reaction, but there is no physical possibility of escape, set this value to **true** to keep the b-radius from being constructed and possible escapes from being assumed. Default is false. It has an effect only for the case of two groups with a reaction.
- **density_field** - dx file containing an external density from microscopy or tomography.
- **atom_weights** - file of atomic weights for each atom name in the atoms file. When there is an external density in **density_field**, the potential of force on each atom is $w\rho$, where ρ is the input density and w is the weight assigned to the atom (usually atomic mass).
- **time_step_tolerances** - This block contains parameters describing limitations on the time step sizes.
  - **minimum_core_dt** - for a system with only rigid cores and frozen chains, this is the smallest possible time step size in picoseconds. Default is zero.
  - **minimum_chain_dt** - This is the smallest possible time step size for a system with unfrozen chains; this typically will be less than **minimum_core_dt**. Default is zero.
  - **minimum_core_reaction_dt** - If a reaction coordinate is small, then the minimum step size given by **minimum_core_dt** can be overridden by the value given by this tag. In other words, the step step still remains larger than that given by **minimum_core_dt** unless the tolerance given by the **reaction** tag is violated. This allows very small time steps near a reaction boundary. The default value is that given by **minimum_core_dt**.
  - **minimum_chain_reaction_dt** - This has the same relation to **minimum_chain_dt**.
  - **constant_dt** - If included, this tells the simulation to use only a constant time step size. For a COFFDROP chain, the

recommended value is 0.05.

- **restraint** - restraint keeping two given atoms from moving more than a certain distance apart
  - **length** - maximum distance between the two atoms
  - **group0** - group of first atom
  - **core0** - core of first atom
  - **chain0** - chain of first atom, if not on core
  - **atom0** - index of first atom
  - **group1,core1,chain1,atom1** - for second atom, same as for first
- **group** - This block describes the rigid cores and chains that make up a *group*.
  - **name** - Each group must have a unique name
  - **core** - This block describes a rigid core. There can be zero or more of these blocks in each group.
    - **name** - Each core must have a unique name
    - **atoms** - the XML file of the core atoms, generated from the PQR file by the program **pqr2xml**
    - **hydro_params** - file with hydrodynamic parameters; output of program **hydro_params**
    - **electric_field** - This block describes the electric field and is optional.
      - **grid** - DX formatted file of electric potential; this is the output from APBS. Can have several.
      - **multipole_field** - file of multipole extension outside grids, output of program **mpole_grid_fit**.
    - **areas** - file of atomic solvent accessible areas (optional). This file comes from the output of the program **get_areas**.
    - **desolvation_field** - This block describes the desolvation field; it is just like the **electric_field** block but does not use the **multipole_field**.
    - **eff_charges** - file with effective charges and lumping information; output of **lumped_charges**. The input into **lumped_charges** can come from other programs such ad **test_charges**, **residue_test_charges**, and **protein_test_charges**.

    - **eff_charges_squared** - file like **eff_charges** above, but with the charges squared. This too comes from **lumped_charges**, but the input to **lumped_charges** first fed through the program **compute_charges_squared**.
    - **copy** - this block allows on to specifify a copy of a previously defined core, but with starting position translated and rotated relative to the parent's starting state. This allows several cores to share the same information, such as electric fields and atoms. When this is used, the above information in the **core** block is ignored, except for **name**.
        - **parent** - the name of the parent core
        - **translation** - the position of the hydrodynamic mobility center relative to that of the parent
        - **rotation** - the 3x3 matrix describing the rotation about the mobility center before translation. (The data in the **translation** and **rotation** blocks together make up the 4x4 affine transformation matrix.)
- **chain** - filename describing a chain. There can be zero or more of these blocks in each group.
- **dummy** - block describing dummy molecule
    - **name** - name of dummy molecule
    - **atoms** - the XML file of the atoms
    - **core** - name of associated core in the same group. These atoms move rigidly with the core.

3.3 Chain File

Unlike the cores, each chain has its own file, because of the larger amount of information.

```
<root>
  <name> name </name>
  <atoms> filename </atoms>
  <areas> filename </areas> # optional

  <constant_dt> real </constant_dt> # optional
```

```
<bonds>
  <bond>
    <atoms> integer integer </atoms>
    <type> name </type>
    <cores> name name </cores> # optional
  </bond> # one or more
</bonds> # optional

<angles>
  <angle>
    <atoms> integer integer integer </atoms>
    <type> name </type>
    <cores> name name name </cores> # optional
  </angle> # one or more
</angles> # optional

<dihedrals>
  <dihedral>
    <atoms> integer integer integer integer </atoms>
    <type> name </type>
    <cores> name name name name </cores> # optional
  </dihedral> # one or more
</dihedrals> # optional

<length_constraints>
  <length_constraint>
    <atoms> integer integer </atoms>
    <length> real </length>
    <cores> name name </cores> # optional
  </length_constraint> # one or more
</length_constraints> # optional

<coplanar_constraints>
  <coplanar_constraint>
    <atoms> integer integer integer integer </atoms>
    <cores> name name name name </cores> # optional
  </coplanar_constraint> # one or more
<coplanar_constraint> # optional
```

```
<excluded_atoms>
  <chain_atom>
    <atom> integer </atom>
    <ex_atoms> integer ... </ex_atoms>
  </chain_atom>
  # zero or more

  <core>
    <name> name </name>
    <core_atom>
      <atom> integer </atom>
      <ex_atoms> integer ... </ex_atoms>
    </core_atom>
    # one or more
  </core>
  # zero or more

</excluded_atoms>
# optional

<one_four_atoms>
  <chain_atom>
    <atom> integer </atom>
    <ex_atoms> integer ... </ex_atoms>
  </chain_atom>
  # zero or more

  <core>
    <name> name </name>
    <core_atom>
      <atom> integer </atom>
      <ex_atoms> integer ... </ex_atoms>
    </core_atom>
    # one or more
  </core>
  # zero or more
```

```
</one_four_atoms>
# optional

<never_frozen> true/false </never_frozen> # optional

<core_interactors>
  <interactor>
    <group> name </group>
    <core> name </core>
    <atom> integer </atom>
    <inner_distance> real </inner_distance>
    <outer_distance> real </outer_distance>
  </interactor>
  # one or more
</core_interactors>
# optional

<chain_interactors>
  <interactor>
    <group> name </group>
    <chain> name </chain>
    <inner_distance> real </inner_distance>
    <outer_distance> real </outer_distance>
  </interactor>
  # one or more
</chain_interactors>
# optional

<copy>
  <parent> name </parent>
  <translation> 3 reals </translation>
  <rotation> 9 reals </rotation>
  <equivalent_cores>
    <match> name name </match> # one or more
  </equivalent_cores>
  # zero or more
</copy>
# optional
```

```
</root>
```

- **name** - every chain has a name
- **atoms** - xml file of atoms; output of **pqr2xml**
- **areas** - file of atomic solvent accessible areas (optional). This file comes from the output of the program **get_areas**. Although a chain atom's SASA will change with the chain's motion, a value used from an initial configuration can still be useful.
- **constant_dt** - Optional. If this chain is unfrozen, then the simulation proceeds with the given timestep size, unless another unfrozen chain has a smaller value given by **constant_dt**.
- **bond** - describes a bond with two atoms.
  - **type** - name of type corresponding to the parameter file, which describes the potential energy function
  - **atoms** - integer indices of each atom
  - **cores** - Although a bond is usually defined for chain atoms, it is possible for a bond to include a core atom. So, the core name is given for its corresponding atom. If the atom is on the chain, an asterisk * is given. If all of the atoms are on the chain, then this block is not needed.
- **angle** - same information as a bond, but with three atoms for a bond angle.
- **dihedral** - same information as a bond or bond angle, but with four atoms for a dihedral angle.
- **length_constraint** - follows the same convention for assigning atoms as the **bond** above.
  - **length** - desired length of constraint
- **coplanar_constraint** - assigns four atoms to lie in a plane; uses same convention for assigning atoms as above.

- **excluded_atoms** - In certain force fields, atoms that are closely connected with each other do not interact according to the non-bonded force field. So, it is possible for each chain atom to have a list of other atoms with which the pair-wise non-bonded forces are not computed.
  - **chain_atom** - describes the excluded atoms for a particular chain atom
    - **atom** - atom on this chain
    - **ex_atoms** - list of atoms on the chaingthat are exluded from interacting with this atom
  - **core** - All core atoms that are excluded from non-bonded interactions with atoms on the chain are listed by core.
    - **name** - name of core
    - **core_atom** - atom of core
      - **atom** - number of core atom
      - **ex_atoms** - list of excluded chain atoms
- **one_four_atoms** - In certain force fields, pairs of atoms that are somewhat more distantly connected, such as those which have three bonds between them, have their non-bonded interactions scaled by a certain amount. This block has the same type of information as **excluded_atoms**.
- **never_frozen** - If this is set to "true", then the chain is never frozen. Default value is true.
- **core_interactors** - Unless the chain is designated as "never frozen", it will remain frozen unless one of its atoms comes within a certain distance of an atom on a core, called the *interactor*.
  - **interactor** - can have more than one
    - **group** - name of group
    - **core** - name of core
    - **atom** - integer index of atom
    - **inner_distance** - If all atoms of the chain are outside of this radius, the chain is not unfrozen until one of the atoms passes within this distance of the interactor atom.

    - **outer_distance** - If at least one atom is within this distance from the interactor atom, the chain is not frozen until all atoms are outside of this distance. It must be larger than the **inner_distance**.

- **chain_interactors** - Similar to **core_interactors**, except the iteractor is another chain, with no atom specified. Instead, about each chain, the smallest possible sphere is constructed that contains its atoms. The distance between the centers of the two enclosing spheres is used to determine freezing and unfreezing.

  - **interactor** - can have more than one
    - **group** - name of group
    - **chain** - name of chain
    - **inner_distance**
    - **outer_distance**

- **copy** - this block allows on to specifify a copy of a previously defined chain, but with starting position translated and rotated relative to the parent's starting state. This allows several chains to share the same information, such as bonds, bond angles, dihedrals, and excluded atoms. When this is used, the above information in the **core** block is ignored, except for **name**.

  - **parent** - the name of the parent chain
  - **translation** - the position of the hydrodynamic mobility center relative to that of the parent
  - **rotation** - the 3x3 matrix describing the rotation about the mobility center before translation. (The data in the **translation** and **rotation** blocks together make up the 4x4 affine transformation matrix.)
  - **equivalent_cores** - The child chain may be associated with several cores which are analogous to but separate from those of the parent chain.
    - **match** - two names; the first is the core associated with this chain, and the second is the one associated with the parent chain.

3.4 Solvent File

This file describes the solvent.

```
<root>
  <debye_length> real </debye_length>
  <dielectric> real </dielectric>
  <relative_viscosity> real </relative_viscosity>
  <kT> real </kT>
  <desolvation_parameter> real </desolvation_parameter>
</root>
```

- **debye_length** - default is infinity
- **dielectric** - relative to vacuum permittivity. Default is 78, that of water.
- **relative_viscosity** - relative to water viscosity. Default is one.
- **kT** - default is one
- **desolvation_parameter** - factor that multiplies the energy of Eq. 6

3.5 Atom file

This is the form for atoms for both cores and chains. The program **pqr2xml** produces this format.

```
<root>
  <residue>
    <name> name </name>
    <number> integer </number>
    <atom>
      <name> name </name>
      <number> integer </number>
      <position> real real real </position>
    </atom>
    # one or more
  </residue>
  # one or more
</root>
```

- **residue** - All atoms are organized into residues
  - **name** - usually the residue type, so does not need to be unique
  - **number** - an integer index; they must be unique and increasing, but there can be gaps.
  - **atom**
    - **name** - usually the atom type; does not need to be unique
    - **number** - an integer index; they must be unique and increasing, but there can be gaps.
    - **position** - x,y,z coordinates

3.6 Reaction File

This describes the reactions and pathways.

```
<root>
  <first_state> string </first_state>
  <reactions>
    <reaction>
      <name> name </name>
      <state_before> name </state_before>
      <state_after> name </state_after>
      <criterion>
        <molecules>
          <molecule0> group_name, substructure_name </molecule0>
          <molecule1> group_name, substructure_name </molecule1>
        </molecules>
        <n_needed> integer </n_needed>
        <pair>
          <atoms> integer integer </atoms>
          <distance> real </distance>
        </pair> # zero or more
        <criterion> ... </criterion> # zero or more
      </criterion>
      # exactly one
    </reaction>
    # one or more
  </reactions>

</root>
```

- **first_state** - name of starting *reaction state*
- **reactions**
  - **reaction**
    - **name** - unique name
    - **state_before** - Each reaction connects two states in one direction
    - **state_after**
    - **criterion** - Each reaction has exactly one criterion at the top
      - **molecules** - Each reacting pair of atoms can be on a different substructure (core, chain, or dummy) in different groups
        - **molecule0** - The core, chain, or dummy is designated by the name of the group and then the name of the core, chain, or dummy
        - **molecule1** - Likewise for the second atom
        - **n_needed** - Number of (pairs + internal criteria) that must be satisfied for reaction
        - **pair** - Pair of atoms, with chains, cores, or dummies designated above. Can have one or more in this block.
          - **atoms** - indices of atoms
          - **distance** - center-to-center distance between atoms below which the pair is satisfied
        - **criterion** - can recursively nest reaction criteria; can have several here

3.7 Forcefield Parameters

All parameters are in one parameter file. Each atom type is designed by a residue type and an atom type within the residue (e.g., an α-carbon within an alanine).

3.7.1 Molecular Mechanics

The pairwise non-bonded parameters are indexed solely by the atom and residue types.

```
<root>
  <units> name </units>
  <one_four_factor> real </one_four_factor>

  <residue>
    <name> residue_type </name>
    <atom>
      <name> atom_type </name>
      <A> parameter </B>
      <B> parameter </B>
      <sigma> parameter </sigma>
      <epsilon> parameter </epsilon>
    </atom>
    # one or more
  </residue>
  # zero or more

  <pair_parameters>
    <parameter>
      <atom0> atom_type </atom0>
      <residue0> residue_type </residue0>
      <atom1> atom_type </atom1>
      <residue1> residue_type </residue1>
      <A> parameter </A>
      <B> parameter </B>
      <sigma> parameter </sigma>
      <epsilon> parameter </epsilon>
    </parameter>
    # one or more
  </pair_parameters>
```

- **units** - units of energy in the input. Can either be “default” (using Browndye units) or “kcal_per_mole”, as is common in many parameter sets. Length units are in Ångstroms.

- **one_four_factor** - factor by which non-bonded energy is multiplied for one-four pair designated in the chain file.
- **residue** - used to give single-atom Lennard-Jones parameters, which then follow the Lorenz-Bertholet combining rules.
  - **name** - type of residue
  - **atom** - type of atom
    - **A** - Lennard-Jones A parameter; units of energy*length^{12}
    - **B** - Lennard-Jones B parameter; units of energy*length^{6}
    - **sigma** - Lennard-Jones σ parameter; units of length (if A and B are not used)
    - **epsilon** Lennard-Jones ϵ parameter; units of energy
- **pair_parameters** - LJ parameters specified by atom pair
  - **parameter** - zero or more
    - **residue0** - type of first residue
    - **atom0** - type of first atom
    - **residue1** - type of second residue
    - **atom1** - type of second atom
    - **A** - Lennard-Jones A parameter; units of energy*length^{12}
    - **B** - Lennard-Jones B parameter; units of energy*length^{6}
    - **sigma** - Lennard-Jones σ parameter; units of length (if A and B are not used)
    - **epsilon** Lennard-Jones ϵ parameter; units of energy

The nonpolar desolvation (or hydrophobic) forces are given in the block

```
<hydrophobic>
  <a> real </a>
  <b> real </b>
  <c> real </c>
  <beta> real </beta>
</hydrophobic>
```

The block, and each of its entries, are optional; the defaults described above are used when the parameter is not given. Parameters a and b are given in Å, c is unitless, and β has units of surface tension, given in terms of Browndye energy and length units.
Each bond, bond angle, and dihedral is designated by its own name. Bonded parameters:

```
<bonds>
  <bond>
    <name> name </name>
    <kr> real </kr>
    <req> real </req>
  </bond>
  # zero or more
</bonds>

<angles>
  <angle>
    <name> name </name>
    <kth> real </kth>
    <theq> real </theq>
  </angle>
  # zero or more
</angles>

<dihedrals>
  <dihedral>
    <name> name </name>
    <mode>
      <n> integer </n>
      <gamma> real </gamma>
      <V> real </V>
    </mode>
    # one to four
  </dihedral>
  # zero or more
</dihedrals>
```

```
</root>
```

- **bonds**
  - **bond** - bond type
    - **name** - name of type
    - **kr** - $k_{b,i}$ in Eq. 5
    - **req** - $r_{eq,i}$ in Eq. 5
- **angles**
  - **angle** - angle type
    - **name** - name of type
    - **kth** - $k_{\theta,j}$ in Eq. 5
    - **theq** - $\theta_{eq,j}$ in Eq. 5
- **dihedrals**
  - **dihedral**
    - **name** - name of type
    - **mode** - inner sum of dihedral term in in Eq. 5
      - **n** - $n_{k,m}$ in Eq. 5
      - **V** - $V_{k,m}$ in Eq. 5
      - **gamma** - $\gamma_{k,m}$ in Eq. 5

3.7.2 Spline-Based

This model of potential energy uses cubic splines fit to evenly spaced data for pair-wise non-bonded interactions, bonds, bond angles, and dihedral angles. The atom and residue type names are mapped to an integer index at the beginning of the file. The index 0 for residue types is reserved as a wildcard; it stands for any residue. The energy function not only depends on the atom and residue types; it also depends on the order along the chain of residues, as in a protein. So, the pair-wise interaction between two β-carbons on adjacent residues may be different than the same interaction between non-adjacent residues.

Bonds and non-bonded pairwise interactions are both listed under the tag $\langle$pairs$\rangle$. The non-bonded interactions, which occur for any pair of atoms with

the designated residue and atom types, are denoted with ⟨order⟩ 0 0 ⟨order⟩" in the "⟨potential⟩" record, while any other numbers in the "order" tag refer to an interaction between specific atoms and are treated as a bond.
Each pair, bond angle, and dihedral has an integer index, rather than a name. These match up with the names in the "type" tags in the chain file. The contents of the "atoms" and "residues" tags map those names to the indices.

```
<top>
  <units> name </units>
  <types>
    <atoms>
      <type>
        <name> name </name>
        <index> integer </index>
      </type>
      # one or more
    </atoms>

    <residues>
      <type>
        <name> name </name>
        <index> integer </index>
      </type>
      # one or more
    </residues>
  </types>

  <pairs>
    <distance> real real </distance>
    <potentials>
      <potential>
        <index> integer </index>
        <residues> integer integer </residues>
        <atoms> integer integer </atoms>
        <orders> integer integer </orders>
        <data> real (two or more) </data>
      </potential>
      # one or more
    </potentials>
  </pairs>

  <bond_angles>
    <angle> real real </angle>
    <potentials>
      <potential>
        <index> integer </index>
        <residues> integer integer integer </residues>
        <atoms> integer integer integer </atoms>
        <orders> integer integer integer </orders>
        <data> real (two or more) </data>
      </potential>
      # one or more
    </potentials>
  </bond_angles>

  <dihedral_angles>
    <angle> real real </angle>
    <potentials>
      <potential>
        <index> integer </index>
        <residues> integer integer integer integer </residues>
        <atoms> integer integer integer integer </atoms>
        <orders> integer integer integer integer </orders>
        <data> real (two or more) </data>
      </potential>
      # one or more
    </potentials>
  </dihedral_angles>

</top>
```

- **distance** - two values are the low and high of the range
- **angle** - same as for distances
- **potential** - for each pair, angle, and dihedral, contains the description of the potential energy and types and orders of atoms and residues

- **residues** - indices of residue types
- **atoms** - indices of atom types
- **orders** - numerical order of residue from N-terminus. For a pair, two 0's denotes a non-bonded pair-wise interaction.
- **data** - values of potential energy at evenly spaced intervals between and including the two values of **distance** or **angle**.

The **atoms**, **residues**, and **orders** tags are not used by the simulation code, since the pair, angle, or dihedral types are already assigned to those structures in the chains. Rather, they are used by other programs like **coff-drop_chain** to help generate chain files.

3.8 Simulation Programs

3.8.1 nam_simulation

In order to run a single-trajectory simulation, the program **nam_simulation** is invoked with the simulation file:

```
nam_simulation simulation.xml
```

The main results are periodicly written to the file specified by the **results** tag in the simulation file. At any time, the 2nd-order reaction rate can be computed by running **compute_rate_constant**:

```
cat results.xml | compute_rate_constant
```

assuming that "results.xml" is the results file. This outputs the estimate of the rates and their 95% confidence intervals to standard output.

3.8.2 process_trajectories

Trajectories can be viewed by processing the trajectory files with the auxiliary programs **process_trajectories** and **vtf_trajectory**. The trajectory files are output in a MIME format within the XML in order to save space, and the completed files include information necessary to quickly select individual trajectories. In addition to the trajectory files themselves, **nam_simulation** also outputs trajectory index files, whose names follow the cooresponding trajectory file. For example, if the trajectory file is "traj2.xml", the index file is "traj2.index.xml". The index file is used in order to avoid plowing through

megabytes of XML in order to find a particular trajectory. Each trajectory is given a number (starting at 0), and each trajectory is divided into one or more numbered subtrajectories. Each time the molecular system completes a reaction, the current subtrajectory terminates, and if the reaction is not final, a new subtrajectory starts. If there is only one reaction, then there is only one subtrajectory. Regardless of the argument of the -nstride flag below, the last conformation before a reaction is always output; this allows the user to see the final bound state, for example.
The program **process_trajectories** takes the following arguments with flags:

- **-traj** : trajectory file from **nam_simulation**
- **-index** : index file
- **-n** : trajectory number in file (starting with 0)
- **-sn** : subtrajectory number (starting with 0); default is 0.
- **-nstride** : only every nstrideth state is output; the default of 1 means that every state is output
- **-srxn** : find subtrajectories resulting from completion of the given reaction (see below)
- **-ucomp** : find subtrajectories starting from the given state but failing to complete

Output to standard output is an XML file, not in MIME format, of a single subtrajectory. In addition, this program lets you select out subtrajectories that have led to a particular reaction. It is called with the -traj and -index flags and their file arguments, and with the flag -srxn with the given reaction name as the argument. This causes it to print out a list, in XML format, of each trajectory and subtrajectory that has led to the reaction. In a similar manner, the program will output a list of subtrajectories starting from a state that lead to an escape; the name of the state is given as the argument to the -uncomp flag.

3.8.3 vtf_trajectory

The output from **process_trajectories** can be processed by vtf_trajectory to generate files that can be viewed. It receives, through standard input, the trajectory file output from **process_trajectories**, and outputs to standard output a VTF file that can be read and viewed with VMD. It can also take this flag as a argument:

- **-pqr**: if included, trajectory is output as concatenated PQR files instead of VTF

3.8.4 rates_of_distances

Often one wants to tune the reaction criteria to match a known rate constant. Instead of using a trial-and-error approach of setting the required distance in a reaction criterion, one can estimate the reaction rate as a function of required distance in one simulation. This is done by running a trajectory, and outputting the minimum reaction distance achieved before escaping for good. The reaction is turned off by setting the required distance to zero, so that all trajectories escape. Then, for each value of the distance, the fraction of trajectories that would have reacted had that distance been the criterion for reaction is computed, along with the corresponding rate constant. This tabulated function can be used to find the required distance to give a desired rate constant. To do this requires four steps. First, set an output file for the minimum reaction distances in the body of the input file by including

```
<min_rxn_dist_file>  dist_file </min_rxn_dist_file>
```

Next, set the reaction distances in the reaction description to zero; this keeps reactions from taking place. Then, run **nam_simulation** as above. Finally, run the program **rates_of_distances**, using as inputs the normal output file of **nam_simulation** and the file named inside the **min_rxn_dist_file** tag:

```
rates_of_distances -res result_file -dist dist_file > output_file
```

The output gives an XML file with gives the rate constant and its 95% confidence interval as a function of distance. If you just want to ouput a plain file for easy plotting, add the flag -plain to the command line.

3.8.5 we_simulation

In order to run a weighted-ensemble simulation, it necessary first to build configuration space bins; the bin information is placed in the file prefix0-prefix1-bins.xml . This is done by running the program**build_bins** :

```
build_bins simulation.xml
```

The number of system copies used is designated by the n-bin-copies tag in the input file. As it runs, reaction coordinate numbers will go scrolling past; they should keepgetting smaller and eventually stop. If that does not happen, i.e., the numbers keep on going, you might need to increase the number of system copies, or it might be that your reaction criterion is unattainable. After the bins are built, the actual weighted-ensemble simulation is run:

```
we_simulation simulation.xml
```

As before, the results are output to the file specified by results . In each row of output numbers, the right-most number is the flux of system copies that escaped without reaction, while the other ones are reactive fluxes. So, even for a rare reaction event, you should at least see small numbers for the reactive fluxes after the system has reacted steady-state. This can be visually examined at any point, and can also be analyzed as above, but with a different program:

```
compute_rate_constant_we < results.xml
```

assuming that **results.xml** is the results file. Because the streams of numbers are autocorrelated, a more sophisticated approach for computing confidence intervals is used [9], and if there are not enough data points, the program **compute_rate_constant_we** will simply refuse to provide an answer and print out a message about autocorrelations not dying away quickly enough. Unlike the single-trajectory method, the weighted-ensemble method cannot handle arbitrary reaction networks; for now, it can only handle nested reaction criteria with the same atom-atom contacts but different required distances. This program can also convert the results file from **we_simulation** into the format output by **nam_simulation**; this is done by including the flag **-nam** in the command.

4 Preprocessing

For most of the programs, typing the program name followed by **-help** will give a description of the function and required arguments.

4.1 pqr2xml

This program converts a PQR file, which describes the collection of charged spheres making up the molecule, to an equivalent XML file (PQRXML file).

The reason for this is that most of the files processed by the UCBD software are in XML format. PQR files can be generated from PDB files using software included with APBS. More information on the PQR format, and the equivalent XML format, can be found in the APBS documention. The pqr2xml program receives the PQR file through standard input, and outputs to standard output. Example:

```
pqr2xml < mol.pqr > mol.xml
```

4.2 make_rxn_pairs

Although the above file can be generated by hand, in the case of two large molecules forming a complex, it is better to have some help at least to generate the pairs. This program takes three files and generates a file listing atom pairs.

- **-mol0** : XML file of the atoms of Molecule 0
- **-mol1** : XML file of the atoms of Molecule 1
- **-ctypes** : file defining possible intermolecular atomic contacts
- **-dist** : distance within which possible pairs are sought
- **-nonred** : removes redunant pairs, as described below

An example is seen in the Makefile of the thrombin example. The two molecule XML files represent the atoms of the molecules in their complexed state, the file addressed by the **-ctypes** flag denotes the types of atoms involved in potential contacts. If any two atoms of a possible pair are within the distance given by the flag **-dist**, then they are output as a reaction pair. The output file pairs.xml contains an XML listing of the reaction pairs. This can be used to make a reaction file.
If the **-nonred flag** is present in the input, then after finding possible pairs, the program eliminates redundant pairs. It groups the pairs according to connectivity, selects the closest bond in each group, and eliminates all other pairs involving the atoms of the closest pair. It then regroups and eliminates repeatedly until no more connected groups are left. Previous studies using SDA used a very similar approach.

The structure of the file **contacts.xml** can enumerate pair possibilities in two ways. One way is to list atom types (which include the residue type) of Molecule 0, and the atom types of Molecule 1, and state that all combinations of atoms from the lists is a possible pair. This is done in the file **protein-protein-contacts.xml.bak** in the tutorial , where all possible hydrogen-bonding pairs are implicitly enumerated. Another way is to explicitly list the pairs. This is the file structure:

```
<contacts>
  <combinations>
    <molecule0>
      <contact>
        <atom> NE2 </atom>
        <residue> HIS </atom>
      </contact>
       ...
    </molecule0>

    <molecule1>
      <contact>
        <atom> OD1 </atom>
        <residue> ASP </atom>
      </contact>
       ...
    </molecule1>

  </combinations>

   ...


  <explicit>
    <pair>
      <contact0>
        <atom> name </atom>
        <residue> name </residue>
      </contact0>
      <contact1>
```

```
        <atom> name </atom>
        <residue> name </residue>
      </contact1>
    </pair>
    ...
  </explicit>

</contacts>
```

The file **protein-protein-contacts.xml.bak** in the thrombin example represents the same rules used by the auxiliary programs in SDA for protein-protein interactions, and thus can be used for systems other than thrombin-thrombomodulin.

4.3 make_rxn_file

This program can make use of the output of **make_rxn_pairs** to create a reaction file.

- **-pairs** : output of **make_rxn_pairs**
- **-nneeded** : number of distinct pairs needed for a reaction
- **-distance** : if -nneeded pairs have a distance less than -distance, then a reaction occurs
- **-rxn** : name of reaction
- **-state_from** : name of state before reaction
- **-state_to** : name of state after reaction
- **-mol0** : followed by two names: the group, and the section (core or chain). This describes the first group.
- **-mol1** : followed by two names: the group, and the section (core or chain). This describes the second group.

An example of usage can be seen in the Makefile of the thrombin example.

4.4 get_areas

This program takes only one argument, the name of the file output by **pqr2xml**, and outputs to standard output an XML file of atomic solvent-accesible areas. The solvent radius used is 1.4Å(later we will add a flag to change that.) This program calls APBS to compute the areas. Right now, it is not incorporated into the calling sequence of **bd_top**, but will be eventually.

4.5 surface_spheres

This program reads a PQRXML file from standard input describing the charged spheres (output from pqr2xml), and outputs an XML file with four lists. The first list is a list of the triangles of the surface spheres; each triangle is a trio of integers, with each integer representing a sphere. These triangles come from an algorithm which is almost identical to that used in Michel Sanner's MSMS. A probe rolls across the molecule surface and touches three spheres at a time as it makes its way. The second list is a list of integers, each representing a surface sphere. The third list is a list of "insiders", or those spheres completely enclosed within a surface sphere. The fourth list is a list of "danglers", or those spheres that hang out into the solvent but could not be picked up by the ball-rolling algorithm. The following input flags are used:

- **-probe_radius** : changes probe radius from default of 1.5
- **-all** : simply include all spheres in surface

Once the surface spheres and their triangles are computed, the program must then distinguish between the interior spheres and the danglers. A point is selected which is guaranteed to outside. Then, for each remaining sphere, a line segment is constructed running from the exterior point to the sphere center. The program counts how many surface triangles are intersected by the line segment (this is done using a log(n) algorithm so that every triangle does not need to be checked). The number of intersections denotes whether the sphere is inside or outside the cage of surface triangles. If the program is unlucky and the line hits a triangle edge, the program will perturb the exterior point slightly and try again.
Example:

```
surface_spheres < mol.xml -probe_radius 1.6 > mol_surface.xml
```

4.6 make_surface_sphere_list

This program generates an XML file (equivalent to PQR) of spheres on the surface. It reads in the surface and dangler spheres from the output of surface_spheres, as well as spheres from the reaction file (output of make_rxn_file) that have not been included by surface_spheres. The following input files are used:

- **-surface** : surface file (output from surface_spheres)
- **-spheres** : XML file of spheres (output from pqr2xml)
- **-group** : group name
- **-core** : core name
- **-rxn** : XML reaction criteria file

Example:

```
make_surface_sphere_list -surface mol_surface.xml -spheres mol.xml
  -group my_group -core my_core -rxn mol_rxn.xml
```

4.7 Programs that generate charges

The following three programs use standard input and output

4.7.1 test_charges

Generates a test charge for each sphere.

4.7.2 residue_test_charges

Generates test charge for each charged residue; position is at center of charge for each residue.

4.7.3 protein_test_charges

Generates test charges for charged residues using the SDA model.

4.8 lumped_charges

The script **lumped_charges** takes as its input the xml file of effective charges from the output of the effective charges programs above and outputs another xml file containing a hierarchical grouping of the charges. The key idea is that if the source of electric field is far away from a group of force centers and you want to compute the force and torque on the group, the group can be compressed into a smaller and faster data structure. Using the technique of Chebyshev interpolation, the group can be converted into a data structure that I'll call a *chebybox*. The chebybox has a rectangular array of 64 positions ($4 \times 4 \times 4$) where the electric potential is evaluated. The resulting force is a linear combination of contributions from each position multiplied by the electric potential evaluated at the position:

$$\mathbf{F} = \sum_{i=1}^{64} V_i \mathbf{f}_i, \tag{9}$$

where V_i is the electric potential evaluated at point i and $\mathbf{f}_i$ is the contribution from point i. The torque is computed from a similar linear combination. Mathematically, the chebybox approximation is exact if the the potential is a cubic function.

Chebyboxes can be nested. If, during the course of a simulation, the field source comes closer to a group of force centers, then the chebybox representing the group might not provide accurate forces and torques, so the group must be split into two groups, each with its own chebybox. For example, near a point charge, the potential cannot be represented by one cubic function over a volume that is large compared to the distance from the charge. The decision is made by computing the ratio of distance of the box center from the field source, to the box diagonal length. If this ratio is below a certain threshhold, the box is divided. Finally, it is not worthwhile to use a chebybox if the number of force centers in a group is much less than 64; it is better to evaluate the force on each center explicitly.

In addition to an outer $4 \times 4 \times 4$ chebybox, the program also generates an outer $3 \times 3 \times 3$ chebybox, which can be used when the molecules are far apart. The following input flags are used:

- **-max** : maximum number of points in box (default 40)
- **-pts** : XML file of charged points;
- **-thr** : distance to diameter ratio for skipping inner boxes (default 2.0);

4.9 mpole_grid_fit

This program computes a multipole fit (out to quadrupole level) to the outer points of the input grid. The fit is done by least squares on the surface of the largest sphere enclosed by the grid. The multipole information is output as an XML file. It also computes the minimum distance from the center at which the field is amost radially symmetric. It can use either an electric field file or a file of charges. The following input flags are used:

- **-dx** : Electric field in DX format
- **-charges** : XML file of molecule charges if there are no arguments for dx
- **-center** : center of fitting sphere (3 numbers). If not included, default is grid center
- **-solvdi** : Solvent dielectric (default 78)
- **-vperm** : vacuum permitivity (default value assuming units of Angstroms, electron charge, and kT (298 K))
- **-debye** : Debye length (default infinity)

4.10 inside_points

This program outputs an XML file representing a rectangular grid of points, each with a 1 or 0 depending on whether the point is inside (1) or outside (0) the molecule. The program reads in the XML sphere data (from **pqr2xml**) and the surface information from **surface_spheres**. For this application, a point is "inside" if it meets at one of two criteria. First of all, if the point's distance from the surface of any surface or dangler atom is less than a certain exclusion distance set by the user, it is considered inside. Second, if the point is inside the cage of triangles formed by the surface spheres, it is considered to be inside the molecule. The lower corner, spacing, and number of points in each direction of the grid are set by the user. The 1's and 0's are output in order, starting from the lower corner, with the x-direction varying most rapidly. If the lower corner or spacing are not specified, reasonable and useful defaults are chosen. The following input flags are used:

- **-spheres** : XML file with sphere data

- **-surface** : XML file with surface data (output from surface_spheres)
- **-exclusion_distance** : additional interior padding around each sphere (default probe radius from surface file)
- **-corner** : lower corner of grid (three numbers)
- **-ngrid** : number of grid points in each direction (default 100 100 100)
- **-spacing** : spacing between grid points (three numbers)
- **-egrid** : use dx file as template for grid corner, size, and spacing instead of previous three arguments

Each grid point is tested for inside-ness in same manner as in **surface_spheres** above. To speed things up, if a point has been found to be inside, the distance to the nearest triangle is found, and all other points within that distance are immediately marked as "inside" as well. If the point is outside, then the distance to the nearest sphere surface is found, and the points within that distance are marked as "outside". This avoids having to find intersecting triangles for most of the grid points.
The big advantage of this algorithm is that it avoids marking interior molecular cavities as exterior. This is essential for efficiently lumping the effective charges together.
Example:

```
inside_points -spheres mol.xml -surface mol_surface.xml
-egrid mol.dx > mol_inside_pts.xml
```

4.11 hydro_params

Computes and outputs hydrodynamic radius [5] and center. Takes output of **inside_points** as input.

4.12 born_integral

This program computes the Born integral on a grid for a molecule core as described in Eq. 6 This integral is computed for all exterior grid points; the geometry is given by the first input. A multipole method is used in order to speed up the execution.

- **-in** : name of input file, which is the output file of 0's and 1's from **inside_points**
- **-vperm** : vacuum permittivity (default in units of $\AA$, ps, kT at 298 K)
- **-ieps** : dielectric of solute (default 4)
- **-oeps** : dielectric of solvent (default 78)
- **-debye** : Debye length (default infinity)
- **-dx** : another dx file can be used as a template for the output, instead of the input to -in
- **-atoms** : instead of using -in to describe the volume, a pqr xml file can be used instead; this gives a grid equivalent to that used in the SDA software package. This is now the default generated from **bd_top**.

4.13 compute_charges_squared

This program takes a file of charges and produces a file of the charges squared. This is used, along with the output of **born_integral**, to compute the desolvation forces.

4.14 COFFDROP Programs

One version of spline-based potentials is COFFDROP from the Elcock group [1, 4]. The following two programs can help with setting up simulations with the COFFDROP force field.

4.14.1 generate_coffdrop_beads

Outputs COFFDROP beads from atoms.

- **-pqr** : file of original atoms in XML format
- **-map** : mapping file from atoms to beads, from Browndye website.
- **-charges** : COFFDROP charge file, from Browndye website
- **-radius** : bead radius in $\AA$. Default is 3.43.

4.14.2 coffdrop_chain

Outputs chain file with bonds, bond angles, dihedrals, and excluded atoms defined from the COFFDROP parameter file. The core information after the −**core** flag has the following format. For each core, the following sequence is given:

```
core_name core_file core_residue0 chain_residue0 (core_residue1 chain_residue1)
```

where the first two tokens are the name and file of the core, and one or two pairs of the core residue number and the number of the attached chain residue. If core_residue0 - 1 exists, then chain_residue must be the lowest residue of the chain. Likewise, chain_residue must be the highest in the chain if core_residue + 1 exists. The chain can be attached to one core at its low or high end, or both ends, or each end can be attached to a different core. The flag −**core** is given only once; if there is a second core, it is appended to the information for the first core.

- **-pqr** : file of atoms in XML format
- **-params** : parameter file, most likely the COFFDROP parameter file found on the Browndye website
- **-conn** : COFFDROP connectivity file, found on the website
- **-name** : name of the chain
- **-core** : information for each attached core; optional
- **-constraints**: no argument; if included, then bonds are replaced with constraints.
- **-frozen**: no argument; if included, then chain can be frozen when far from an interactor.
- **-dt**: time step size. If included, includes the value into the **constant_dt** tag in the output.

4.15 mrc2dx

Converts mrc-formatted file to dx format. Uses standard input and output

- **-factor**: multiplicative factor (default 1)
- **-exp**: outputs factor*exp(density) rather than factor*density; default is false

4.16 scale_interactions

Reads in the COFFDROP file and scales the negative values of the non-bonded interactions. Outputs to standard output.

- **-in**: input COFFDROP file
- **-scale**: scale factor (default 1)

5 Flow of Control

Until I include an updated version of **bd_top** from the previous version of Browndye, which will do most of this automatically, users will have to go through the following steps.
Starting files:

- PQR files of the cores
- PQR files of the chains
- Reaction file
- Electrostatic grid files from APBS for the cores
- Simulation XML file

Steps:

- Run **pqr2xml** to convert PQR files (chains and cores) to XML files
- Run **surface_spheres** to obtain a surface description file for each core
- Run **make_surface_sphere_list** to convert the surface description file from **surface_spheres** into a file with just surface atoms.

- Run **inside_points** for each core to obtain a description of the core interior, from the output of **make_surface_sphere_list**.
- Run **hydro_params** for each core to get the hydrodynamic parameters, from the output of **inside_points**.
- If using desolvation forces, run **born_integral** on the output of **inside_points**.
- Run one of the charge producing programs above (**test_charges**,**residue_test_charges**, or **protein_test_charges**) on the output of **pqr2xml** to get a file of charges.
- Run **lumped_charges** on the file of charges to get the lumped charges.
- If using desolvation forces, run **compute_charges_squared** on the charge file above to get a file of charges squared, and run **lumped_charges** on that to get the lumped quantities.
- Run **mpole_grid_fit** on the outermost electrostatic grid of each core to get the off-grid multipole model.
- If using the spline-based COFFDROP model, run **generate_coffdrop_beads** on the core and chain atoms files to get "atom" files with the coarse-grained beads.
- Build chain files.

For now, the chain files need to be built by hand if the molecular mechanics model is being used; I hope to have additional programs to help the process for suitable models. The **coffdrop_chain** program can be used to build chains that use the COFFDROP model, but for now it does not include the necessary steps to attach a chain or loop to a core; that still needs to be done by hand. That ability will be added soon, however.

6 Other Brownian Dynamics Packages

These are the ones I can think of right now; please let me know if I'm missing any other important or interesting ones.

- SDA `https://mcm.h-its.org/sda/doc/doc_sda7/index.html`

- GeomBD `http://chemcha-gpu0.ucr.edu/software/`
- MacroDox `http://www.cae.tntech.edu/~snorthrup/macrodox/macrodox.html`
- Atomic Resolution Brownian Dynamics `http://bionano.physics.illinois.edu/arbd`
- BD_Box `https://bionano.cent.uw.edu.pl/Software/BD_BOX`
- ReaDDy `https://readdy.github.io/`
- BDPack `http://amir-saadat.github.io/BDpack/`
- BrownMove `https://www-cbi.cs.uni-saarland.de/services/the-brownmove-many-par`
- UHBD (The grandma of them all! If anyone knows of a good link, please let me know; otherwise, I might set up another website.)

7 References

References

[1] Casey T. Andrews and Adrian H. Elcock. COFFDROP: A Coarse-Grained Nonbonded Force Field for Proteins Derived from All-Atom Explicit-Solvent Molecular Dynamics Simulations of Amino Acids. *Journal of Chemical Theory and Computation*, 10(11):5178–5194, NOV 2014.

[2] Adrian H. Elcock. Molecule-Centered Method for Accelerating the Calculation of Hydrodynamic Interactions in Brownian Dynamics Simulations Containing Many Flexible Biomolecules. *Journal of Chemical Theory and Computation*, 9(7):3224–3239, JUL 2013.

[3] D. L. Ermak and J. A. McCammon. Brownian dynamics with hydrodynamic interactions. *J. Chem. Phys.*, 69(4):1352–1360, 1978.

[4] Tamara Frembgen-Kesner, Casey T. Andrews, Shuxiang Li, Nguyet Anh Ngo, Scott A. Shubert, Aakash Jain, Oluwatoni J. Olayiwola, Mitch R. Weishaar, and Adrian H. Elcock. Parametrization of Backbone Flexibility in a Coarse-Grained Force Field for Proteins (COFFDROP) Derived

from All-Atom Explicit-Solvent Molecular Dynamics Simulations of All Possible Two-Residue Peptides. *Journal of Chemical Theory and Computation*, 11(5):2341–2354, MAY 2015.

[5] S Hansen. Translational friction coefficients for cylinders of arbitrary axial ratios estimated by monte carlo simulation. *Journal of Chemical Physics*, 121(18):9111–9115, Nov 8 2004.

[6] G. A. Huber and S. Kim. Weighted-ensemble Brownian dynamics simulations for protein association reactions. *Biophysical Journal*, 70:97–110, 1996.

[7] G. A. Huber and J. A. McCammon. Browndye: a software package for Brownian dynamics. *Computer Physics Communications*, 181:1896–1905, 2010.

[8] S. Northrup, S. A. Allison, and J. A. McCammon. Brownian dynamics simulation of diffusion-influenced biomolecular reactions. *J. Chem. Phys.*, 80:1517–1524, 1984.

[9] DN Politis and JP Romano. The stationary bootstrap. *Journal of The American Statistical Association*, 89(428):1303–1313, Dec 1994.

[10] Leonardo G. Trabuco, Elizabeth Villa, Kakoli Mitra, Joachim Frank, and Klaus Schulten. Flexible fitting of atomic structures into electron microscopy maps using molecular dynamics. *Structure*, 16(5):673–683, MAY 2008.