

The Proteus software for computational protein design: a preliminary user's manual

last updated: August 30, 2013

Thomas Simonson

Department of Biology, Ecole Polytechnique, Palaiseau, France.

thomas.simonson@polytechnique.fr

Proteus is available from <http://biology.polytechnique.fr/biocomputing>

Acknowledgements

Proteus is described in the following article:

Thomas Simonson, Thomas Gaillard, David Mignon, Marcel Schmidt am Busch, Anne Lopes, Najette Amara, Savvas Polydorides, Audrey Sedano, Karen Druart, and Georgios Archontis (2013) *J. Comp. Chem.*, in press; doi: 10.1002/jcc.23418. Computational protein design: the Proteus software and selected applications.

An earlier version was described in:

M. Schmidt am Busch, A. Lopes, D. Mignon & T. Simonson (2008) *J. Comp. Chem.*, 29:1092-1102. Computational protein design: software implementation, parameter optimization, and performance of a simple model.

M. Schmidt am Busch, A. Lopes, D. Mignon, T. Gaillard & T. Simonson (2012) In *Quantum Simulations of Materials and Biological Systems* (editors: J. Zeng, R. Q. Zhang, H. Treutlein), Springer Science, Dordrecht, pages 121–140. The Inverse Protein Folding Problem: Protein Design and Structure Prediction in the Genomic Era.

Thomas Gaillard and David Mignon contributed to this documentation. In addition to the authors listed above, additional contributions, helpful discussions or suggestions were made by Alfonso Jaramillo, Christine Bathelt, Alexey Aleksandrov, Seydou Traoré, and Jialin Liu. The first version of the proteus C code was based on an earlier program by L. Wernisch.

Contents

1	Overview	3
2	Directory structure and files	4
2.1	XPLOR program	4
2.2	Source directories for Proteus	4
2.3	User directories for a Proteus application	5
3	Using Proteus for a typical protein or protein:ligand system	6
3.1	System preparation for XPLOR	6
3.2	CPD setup: files to edit	6
3.3	The energy matrix	7
4	Exploring sequence/rotamer space with proteus	10
5	Rotamer library organization	12
6	A test case: the chignolin decapeptide	13
7	XPLOR code listing for selected scripts	14
7.1	lib/sele.str and lib/parameters.str	14
7.2	inp/matrixI.inp	18
7.3	inp/matrixIJ.inp	24
8	Implementation of Generalized Born solvent models in XPLOR	33
8.1	Introduction	33
8.2	Theory	33
8.2.1	GB energy	33
8.2.2	Calculation of forces	35
8.2.3	Pairs of interacting groups	37
8.2.4	Crystal symmetry	38
8.3	Syntax	38
8.3.1	GB energy terms	38
8.3.2	Setting the GB options	38
8.3.3	Setting up atomic volumes for GB	39

8.3.4	Examples	40
8.3.5	Molecular dynamics with GB/HCT	40

1 Overview

Proteus has four components:

1. the molecular simulation program XPLOR [1], with local modifications;
2. a sophisticated set of scripts, written in the XPLOR scripting language [1, 2], that control the calculation of an energy matrix for the system of interest [3];
3. a C program, “proteus” (with a lowercase “p”) for exploring the space of sequences and conformations using various search algorithms, including a Monte Carlo method;
4. a collection of perl and shell scripts that automate various steps.

To use this manual, the reader should first carefully read the Proteus article (Simonson et al, J Comp Chem, 2013) [4], which includes details on the theoretical methods and the energy function. In addition, the reader should have a copy of the XPLOR manual, and preferably some familiarity with either XPLOR or a similar program, such as CNS [2, 5] or Charmm [6]. The XPLOR manual is available online (as of today) at:

<http://www.pasteur.fr/recherche/unites/Binfs/xplor/manual>

or:

<http://www.csb.yale.edu/userguides/datamanip/xplor/xplorman/htmlman.html>.

A slightly older version (probably sufficient) is provided as part of the Proteus distribution. The manual can also be purchased in book form (A. Brunger; Yale University Press).

We assume the reader is familiar with Unix. The distribution files will work best in a linux environment with an Intel processor and an Intel compiler, although compilation should not be necessary for Intel-based machines, and using Gnu compilers should not be difficult.

Here, we first describe the directory structure in the Proteus distribution and the main files that are used in applications.

Second, we describe the steps in a typical application: system preparation, energy matrix calculation, searching sequence/conformation space, postprocessing and analysis.

Third, we include a description of the logical structure of the rotamer library.

Fourth, we describe and comment a test case provided with the distribution (the Chignolin decapeptide).

Fifth, we include a few of the main script files explicitly.

Last, we include detailed documentation for the generalized Born (GB) model implemented in XPLOR [7, 8].

2 Directory structure and files

2.1 XPLOR program

The organization of the XPLOR files is described in detail in the XPLOR manual. The version distributed with Proteus has the same structure, with local modifications to individual source code files. The top XPLOR directory could be something like `/usr/local/xplor3.8` or `/usr/local/Proteus/xplor3.8`. It is normally defined by the environment variable `$XPLOR`. The main XPLOR subdirectories are the following:

- **`$XPLOR/source`**: source code
- **`$XPLOR/toppar`**: topology and parameter files
- **`$XPLOR/intel64`**: machine and compiler-specific source code; useful shell scripts
- **`$XPLOR/objects_intel64`**: object files and `xplor.exe` binary
- **`$XPLOR/test3.1`**: example scripts

These subdirectories are normally defined as environment variables: `$SOURCE`, `$TOPPAR`, `$COMS`, `$OBJ`, `$TEST`. With `csh`, these definitions can be set by sourcing the file `$XPLOR/intel64/ulogin.com`. With `bash`, equivalent commands can be used. Below, we use these environment variables as shorthands for the corresponding directories.

2.2 Source directories for Proteus

Alongside the top XPLOR directory, we define a top Proteus source directory, say `$CPD`. This could be something like `/usr/local/Proteus`. The main CPD subdirectories are:

- **`$CPD/inp`**: XPLOR scripts for system setup and energy matrix calculation
- **`$CPD/lib`**: XPLOR macros or “stream files” for system setup and energy matrix calculation
- **`$CPD/bin`**: perl and shell scripts
- **`$CPD/rotamers`**: files that define the protein rotamer libraries
- **`$CPD/rotamers_other`**: some non-protein rotamer definitions
- **`$CPD/doc`**: documentation files, including this manual
- **`$CPD/testcase`**: a simple testcase

The most important files are described in the next sections.

2.3 User directories for a Proteus application

For a user running a given application, we define a top project directory, say \$PROJ. This could be something like /home/smith/crk (Crk is a small SH3 protein). The subdirectory setup is partly imposed by the software, especially the matrix calculation. A typical setup would be the following:

- **\$PROJ/build:** initial system setup for XPLOr
- **\$PROJ/lib (or \$MYLIB):** local copy of the XPLOr stream files that define the main parameters for the calculation; edit as needed, especially parameters.str, sele.str
- **\$PROJ/matrix:** top directory for the energy matrix calculation; includes shell scripts to run the calculation
- **\$PROJ/matrix/dat:** the actual matrix files will be written here
- **\$PROJ/matrix/out:** XPLOr output files from the calculation are written here
- **\$PROJ/matrix/err:** XPLOr error messages are collected here
- **\$PROJ/matrix/local:** intermediate files are stored here
- **\$PROJ/matrix/local/Bsolv:** atomic solvation radii (with GB solvent) are written here in bsolv.pdb
- **\$PROJ/matrix/local/Chis:** files defining “native” rotamers, when used
- **\$PROJ/matrix/local/EnrFltr:** files defining the rotamers that have passed an energy filter test
- **\$PROJ/matrix/local/Mut:** position-specific mutation spaces; can be edited manually if needed
- **\$PROJ/matrix/local/Nbrot:** information on the number of rotamers at each position
- **\$PROJ/matrix/local/Rota:** the actual 3D sidechain structures for each rotamer, positioned on the protein backbone
- **\$PROJ/proteus:** directory for the Monte Carlo simulations
- **\$PROJ/reconstruct:** directory for 3D structure rebuilding and postprocessing

The \$PROJ/lib subdirectory should be defined as the environment variable **\$MYLIB**.

3 Using Proteus for a typical protein or protein:ligand system

3.1 System preparation for XPLOR

Protein setup starts from a PDB file, usually edited so that the atom names conform to the conventions of the force field that will be employed. The main model parameters are set by editing a single file, **\$MYLIB/parameters.str**, which is written in the XPLOR command language and where the user sets flags for the choice of force field, solvent model, dielectric constant, and so on. XPLOR is run using a script **\$PROJ/build/build.inp** (which reads parameters.str):

```
xplor < build.inp > build.out
```

The main result is a “Protein Structure File” or PSF, say `allh_protein.psf`, which describes the “topology” or “2D” chemical structure of the protein (sequence, atom types, atomic charges, covalent structure) [1, 6]. If a single, inactive ligand is to be used, it can be created in `build.inp` and written to `allh_protein.psf`. If several ligands are to be used, with mutations to exchange them, one “wildtype” ligand should be included in the `build.inp` step, and the others should be created separately by the user, each with its own PSF file. Each one must be compatible with the force field employed.

3.2 CPD setup: files to edit

For the system build, only **\$MYLIB/parameters.str** had to be edited. For the following steps, a series of files in **\$MYLIB** should be carefully inspected and modified as needed:

- **parameters.str**: sets the force field, solvent model, dielectric constant, and other parameters
- **sele.str**: defines the groups that are “active” (they can mutate), “inactive” (they can’t mutate but are flexible), or “frozen” (their position is fixed)
- **mutation_space.dat**: defines the possible amino acid types for active sidechains; additional restrictions can be applied later on a position-by-position basis (see below)
- **phia.str**: sets the atomic surface energy coefficients
- **refener.str**: sets the “reference” or unfolded state energies for each amino acid type; these should be chosen consistently with the other energy parameters
- **oneletterLIGA.str**: define one letter codes for any active or inactive ligands
- **other parameters** are set in the **\$CPD/lib** stream files, including `nb.str`, `toppar.str`, `oneletter.str`, but do not usually need to be changed.

3.3 The energy matrix

A flowchart for the entire calculation is shown in Fig. 1.

System preparation for CPD The initial “build” step above was a generic preparation for XPLOR. A second, more complex step starts from the generic build and prepares the system specifically for a CPD calculation. It can be run using a bash script, `$PROJ/matrix/setup.sh`. It uses a second XPLOR script, `$CPD/inp/setup.inp`. The user edits the file `$MYLIB/sele.str` to indicate which residues will be active (they mutate), inactive (they are flexible but don’t mutate), or frozen. The main task performed by `setup.inp` is to modify the active residues, by grafting (“patching”) all possible sidechain types onto their backbone C_α . The resulting residues are referred to as “giant” residues. For an active *ligand*, the corresponding operation is to simply read in preexisting PSF files for the different ligand types, so that they coexist in the system. The allowed residue types are listed in a file `$MYLIB/mutation_space.dat`, and might include all amino acid types, or a smaller set for some applications (like pK_a calculations). This file can be edited, but restrictions on possible mutations are more readily applied at later stages. A secondary task performed by `setup.inp` is to analyze the solvent accessibility of each residue; this is needed later to apply a pairwise additivity correction to the surface energy term. Another secondary task, when a GB solvent is used, is to precompute the GB solvation radii for each backbone atom, using a “Native Environment” approximation for the rest of the system (sidechains, ligands) [4]. The output from this step includes a PSF file (`setup.psf`, written to the current directory, usually `$PROJ/matrix`) including giant residues and possibly multiple versions of one or more ligands. A PDB file (`setup.pdb`) is produced, where the giant residues have unassigned sidechain coordinates, except for the wildtype sidechain, and positions are tagged as active/inactive/frozen. A second PDB file `bsolv.pdb` is produced containing the GB solvation radii, and stored in `$PROJ/matrix/local/Bsolv`.

Diagonal matrix elements The energy matrix is computed with XPLOR, in two main steps, executed by two shell scripts. A flowchart for the calculations is shown in Fig. 1. The first step (shell script `runI.sh`; XPLOR script `matrixI.inp`) computes several quantities for each position in the system. The calculations are done as follows. For each active or inactive amino acid position i , we loop over its possible types and rotamers. Each rotamer is placed by superimposing a library rotamer structure onto the protein backbone, based on the N, C_α , C_β , and C atoms; the rotamer coordinates replace the original ones for atoms beyond C_β . Sidechain coordinates of position i beyond C_β are then energy minimized for $N_{\min} = 15$ steps, using the conjugate gradient algorithm. Harmonic restraints are applied to the dihedrals of i , with a force constant of 200 kcal/mol/rad² and a tolerance range of $\pm 5^\circ$ around the ideal, rotamer angle. The rest of the protein is kept fixed; The only interactions considered during the minimization are those of sidechain i with itself and with the protein backbone (“extended” to include any frozen residues). At this point, we save the rotamer coordinates to a file (in `$PROJ/matrix/local/Rota`). We also compute the solvation radii of the sidechain atoms and store them in `bsolv.pdb`.

Finally, we compute the molecular mechanics energy with the considered interactions,

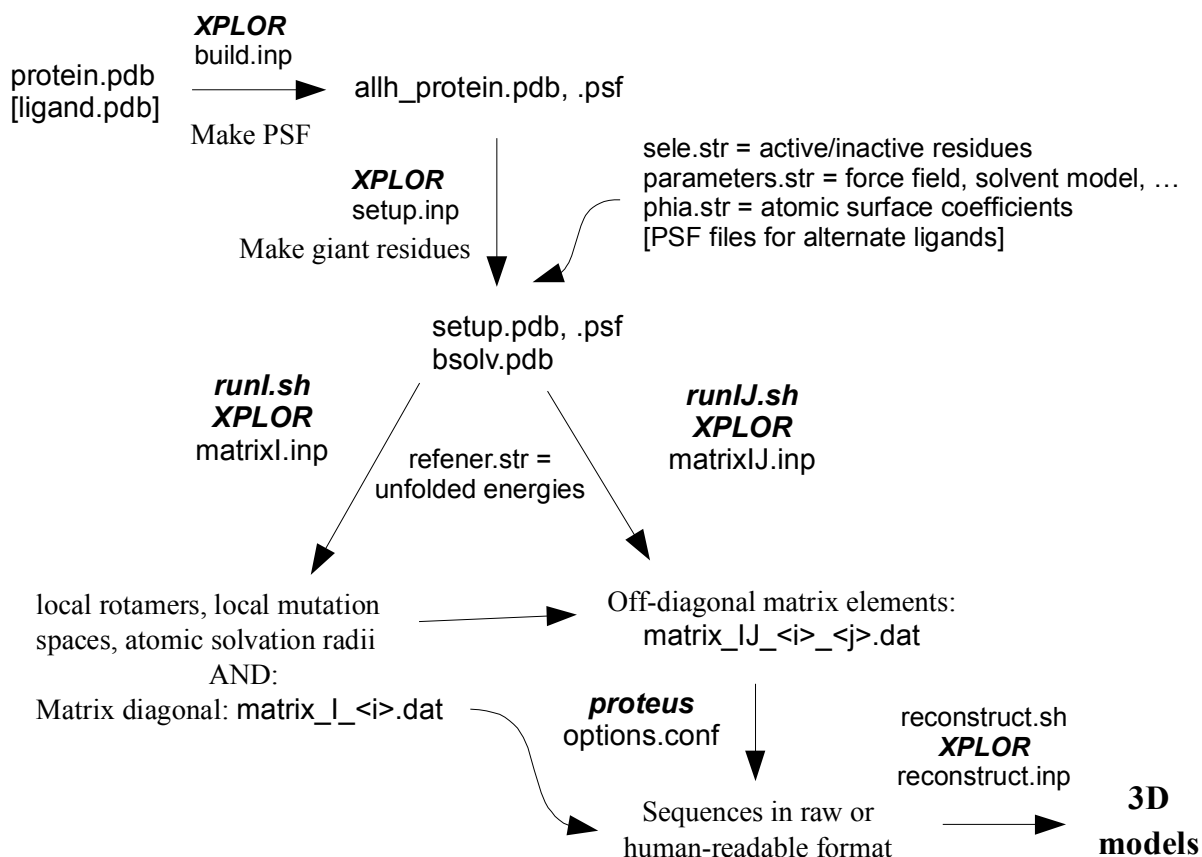


Figure 1: Flow chart for the energy matrix and sequence generation

including the GB implicit solvent term (but omitting the dihedral restraints). A surface energy contribution is also computed, as follows. We estimate a “contact” surface between sidechain i and its environment: the solvent accessible surface area of sidechain i alone, minus the area of sidechain i buried by the (extended) backbone, minus the area of the extended backbone buried by sidechain i . The atomic contributions to this contact area are multiplied by type-specific coefficients to yield a solvation energy term. An unfolded state contribution E_X is subtracted [4], to obtain the ii diagonal element of the energy matrix, which is written to a file, `matrixI_<i>.dat`. The unfolded contributions E_X are read from the file `$MYLIB/refener.str`.

Off-diagonal elements For the off-diagonal matrix elements ij , the calculations are controlled by a shell script `runIJ.sh` and an XPLOR command script `matrixIJ.inp`. The XPLOR script loops over all active and inactive positions i and all their types and rotamers, as did `matrixI.inp` above. For each one, we consider all positions j whose C_β is less than 15 Å from that of i . A library rotamer is placed at position j (by reading the local rotamer file created above). If the minimum distance between the atoms of sidechains i and j is more than 12 Å, we discard this j rotamer and move on to the next one. Otherwise, energy minimization of sidechain j beyond C_β is done, in two steps: first, in the context of the extended backbone (as with sidechain i), then in the context of the extended backbone plus sidechain i . In this second step, both sidechains i and j beyond C_β are allowed to move with dihedral restraints. The interactions considered are those of sidechain i with itself and the extended backbone,

sidechain j with itself and the extended backbone, and sidechain i with sidechain j . The final molecular mechanics energy and ij surface energy are computed (omitting the dihedral restraint energy) and written to the matrix file `matrixIJ_ij.dat`. The procedures for the ii and ij matrix elements are schematized below:

Procedure to compute ii diagonal matrix element

```

1  foreach variable position i {
2    get mutation space for position i
3    if nativerot then handle position i native rotamers
4    if gb then compute position i backbone solvation radii
5    foreach amino acid type ti in mutation space i {
6      get number of rotamers for amino acid ti
7      foreach corresponding rotamer ri {
8        position rotamer ri
9        if gb then compute rotamer ri solvation radii
10       minimize rotamer ri
11       if gb then update rotamer ri solvation radii
12       calculate ii energy matrix element
13       write coordinates of rotamer ri
14     }}}

```

Procedure to compute ij off-diagonal matrix element

```

1  foreach variable position i {
2    get mutation space for position i
3    foreach amino acid type ti in mutation space i {
4      get rotamer space for type ti at position i
5      foreach corresponding rotamer ri {
6        read coordinates of rotamer ri
7        foreach variable position j < i {
8          if ij Cbeta distance below threshold then
9            get mutation space for position j
10           foreach amino acid type tj in mutation space j {
11             get rotamer space for type tj at position j
12             foreach corresponding rotamer rj {
13               read coordinates of rotamer rj
14               if min dist sidei / sidej < 12 A then
15                 if min dist sidei / sidej < 3 A then
16                   minimize rotamers ri, rj
17                 end
18               calculate matrix element ij
19             end
16             end
17           }}}
18         end
19       end
20     }}}
21   end
22 }}}}

```

The entire scripts `matrixI.inp` and `matrixIJ.inp` are listed further on.

4 Exploring sequence/rotamer space with proteus

With the matrix in place, the sequence/rotamer exploration is done with a C program, called proteus (not to be confused with the entire Proteus package). A single command file controls the calculation, with a simple XML format and flexible commands. The main parameters the user can set are listed in Table 1: exploration method, number of steps, choice of the starting sequence/structure, restrictions on sequence/rotamer space, and so on. An example is shown in Fig. 2. Sequences are output in the form of lists of rotamers, along with their folding energies. Rotamers are numbered using the internal proteus numbering, which identifies both amino acid type and rotamer.

Conversion to a more verbose, human-readable format is done by proteus in a separate step. The verbose, or “rich” format includes residue types, numbers, and rotamer numbers (with the numbering of the rotamer library). From each sequence in this format, a perl script can produce a PDB file with the wildtype backbone coordinates (extended to include any frozen residues or ligands), and with the rotamer numbers in the B-factor field. An XPLOR script, reconstruct.inp, can then produce the full PDB structure, sidechains included, with or without some additional overall energy minimization. A series of perl scripts is also available to compute sequence properties, such as similarity to a reference alignment.

Table 1: Possible commands in the proteus command file

XML tag name	Description
Mode	heuristic, MC, mean field or POSTPROCESS
Group_definition	group interaction energies are the basic elements of the energy function
Optimization_Configuration	definition of the energy function
Space_Constraints	restrict possible states or force two residues to have the same type
Seq_Input_File	input file with the starting rotamers
Trajectory_Length	length of an MC trajectory
Trajectory_Number	number of MC trajectories
Cycle_Number	number of heuristic cycles
Sequence_Pass_Number	maximum passes over the structure per heuristic cycle
Lambda_Parameter	Mean Field relaxation parameter
Temperature	for Mean Field or Monte Carlo
Rseed_Definition	seed for the random number generator
Surf_Ener_Factor	energy parameter
Dielectric_Constant	energy parameter
Rot_Proba	probability to have a rotamer move at each MC step
Rot_Rot_Proba	probability to have a move with two rotamer changes
Mut_Proba	move probability
Mut_Mut_Proba	move probability
Mut_Rot_Proba	move probability
Neighbor_Threshold	energy threshold, defines positions that can move together
Seq_Output_File	output file
Energy_Output_File	output file

```

#
# proteus command file for an MC run with the Crk SH3 domain and a bound 9-peptide
#
<Mode> MONTECARLO </Mode>          # use MC for sequence/structure exploration
<Energy_Directory>
  ../matrix                          # location of the energy matrix
</Energy_Directory>
<Temperature> 0.6 </Temperature>    # in kT units
<Trajectory_Number> 1 </Trajectory_Number> # number of MC runs
<Trajectory_Length>
  100000000                          # number of MC steps per run
  # (careful: the output file will be big...)
</Trajectory_Length>
<Seq_Output_File>
  prod.seq                            # output file for sequences/structures/energies
</Seq_Output_File>
<Group_Definition>
  pept 1-9                            # define the peptide and protein (by residue numbers)
  prot 134-190                        # as two distinct groups, called pept and prot
</Group_Definition>
<Optimization_Configuration> # use all interactions (done by default anyway);
m(prot+pept+prot~pept)             # prot~pept represents the inter-group contribution
</Optimization_Configuration>
<Space_Constraints>
  4 ALA
  5 LEU                                # fix the peptide ligand's sequence; peptide
  8 LYS                                # residues 1-3, 6-7 are frozen already (prolines)
  9 LYS
</Space_Constraints>
<Seq_Input_File>
  starting.seq                        # choose the initial sequence/structure
</Seq_Input_File>                  # (eg, the endpoint of a previous run)

```

Figure 2: Proteus command file for an MC simulation

5 Rotamer library organization

The protein rotamer libraries are stored in \$CPD/rotamer. The library recommended for use with the Amber ff99SB force field is in \$CPD/rotamer/ff99SB/Tuffery95_bbind_H. There are five subdirectories: Rota, Chis, Nbrot, Pick, and Rest. Rota contains 3D coordinates for each rotamer; the others contain rotamer information in the form of small XPLOR stream files. For example, the files corresponding to the serine (SER) sidechain are:

subdirectory	files for SER sidechains	content
Rota	SER_1.pdb,..., SER_9.pdb	3D coordinates for each rotamer
Chis	SER_1.dat,..., SER_9.dat	Torsion angle values
Nbrot	SER.dat	Number of sidechain torsions
Pick	SER.dat	Stream file to extract the sidechain torsion values for a current 3D structure
Rest	SER.dat	Stream file to apply dihedral restraints corresponding to a current rotamer

Specifically, the files look like this:

Chis/SER_1.dat:

```
1 eval ($chi1 = 62.0)
2 eval ($chi2 = -60.0)
```

Nbrot/SER.dat:

```
1 eval ($nbrot = 9)
```

Pick/SER.inp:

```
1 pick dihe (resid $resid and resn SER and name n)
2           (resid $resid and resn SER and name ca)
3           (resid $resid and resn SER and name cb)
4           (resid $resid and resn SER and name og) geom
5 eval ($chi1 = $result)
6 pick dihe (resid $resid and resn SER and name ca)
7           (resid $resid and resn SER and name cb)
8           (resid $resid and resn SER and name og)
9           (resid $resid and resn SER and name hg) geom
10 eval ($chi2 = $result)
```

Rest/SER.dat:

```
1 assign (resid $resid and resn SER and name n)
2        (resid $resid and resn SER and name ca)
3        (resid $resid and resn SER and name cb)
4        (resid $resid and resn SER and name og) $dihecons $chi1 $diherange 2
5 assign (resid $resid and resn SER and name ca)
6        (resid $resid and resn SER and name cb)
7        (resid $resid and resn SER and name og)
8        (resid $resid and resn SER and name hg) $dihecons $chi2 $diherange 2
```

6 A test case: the chignolin decapeptide

This is a 10 residue peptide that forms a two-stranded β sheet. Additional test cases will be added later. We refer to the test directory (chignolin_ff99SB_gbsa/) as \$TEST. Model parameters are assigned in \$TEST/lib, also defined as \$MYLIB. The force field, solvent model, and many other parameters are set in parameters.str. We use the Amber ff99SB force field with a simple but well-optimized GB variant we call GB/HCT [7, 8]. In sele.str, we set residue 5 to be inactive and residue 3 to be active. The others are frozen: they do not mutate or explore rotamers. The individual steps are listed below, with a few comments. In the Proteus distribution, most but not all of the output files have been left in place. The sequence/rotamer exploration is done by a few very short Monte Carlo runs, at a fairly high temperature (kT = 1 kcal/mol), for illustration. This leads to just four different sequences, with the residue 5 sidechain types His, Thr, Cys, and Met (see the files \$TEST/proteus/proteus.rich and proteus_sequences.dat). Some of the designed structures are in \$TEST/reconstruction/scr (100 structures for each of the four sequences). Some energy statistics information is temporarily disabled due to recent changes in the sequence file formats.

\$TEST		
subdirectory	main files	comments
build	build.inp, model.pdb, run.sh, build.out	the chain termini are unpatched (historical reasons), with dangling NH and CO
lib	sele.str, parameters.str, phia.str, refener.str, mutation_space.dat	XPLOR stream files, which set most of the model parameters; mutation space and reference energies are needed for active position 5
matrix	setup.sh, runI.sh, runIJ.sh, dat/ local/	the matrix calculation is run from here; the matrix files are written in dat/
matrix/local subdirectories	Bsolv/ Chis/ EnrFltr/ Mut/ Nbrot/ Rota/	these contain the position-specific information on allowed mutations, rotamers, and GB radii
proteus	run.sh, proteus.seq, proteus.MONTECARLO.conf, exploration/	run.sh does everything; xxx.conf files are proteus command files; xxx.seq are designed sequences in raw "rotamer" format
reconstruction	run.sh, scr/	Generate 3D structures from the rotamer information in proteus.seq; run.sh does everything, using \$CPD/inp/reconstruct.inp; PDB files are in scr/

7 XPLOR code listing for selected scripts

The scripts explicitly included here are sele.str, parameters.str, matrixI.inp, matrixIJ.inp.

7.1 lib/sele.str and lib/parameters.str

sele.str

```
1 remark
2 remark Define selections
3 remark
4
5 ! Define inactive residues
6 vector ident (store1) (not (resn GLY or resn CYX or resn PRO))
7
8 ! Define active residues
9 vector ident (store2) (not all)
10
11 ! Ligand center atom
12 vector ident (store3) (segid LIGA and tag)
13
14 ! Ligand fitting atoms
15 vector ident (store5) (segid LIGA)
16
17 ! Protein fitting atoms
18 vector ident (store6) ((not segid LIGA) and (name N or name CA or name CB or name C))
19
20 ! Ligand backbone
21 vector ident (store7) (segid LIGA)
22
23 ! Protein backbone
24 vector ident (store8) ((not segid LIGA) and (resn GLY or resn CYX or resn PRO or
25     (name N or name H or name HN or name CA or name HA or name C or
26     name O or name OXT or name OT+ or name H+ or name HT+)))
27
28 ! Whole backbone
29 vector ident (store9) ((not (store1 or store2 or segid LIGA)) or store7 or store8)
```

parameters.str

```
1 remarks
2 remarks Define parameters
3 remarks
4
5 ! Force field (toph19 or ff99SB)
6 eval ($ff = "ff99SB")
7
8 ! ===== Generalized Born model =====
9 ! GB flag
10 eval ($gb = 1)
11
```

```

12 ! GB HCT solvation model
13 eval ($gbhct = 1)
14
15 ! GB HCT offset
16 eval ($offset = 0.00)
17
18 ! GB HCT lambda
19 eval ($lambda = 1.0)
20
21 ! GB ACE solvation model
22 eval ($gbace = 0)
23
24 ! GB ACE smooth
25 eval ($smooth = 1.3)
26
27 ! GB HCT/ACE solvent dielectric constant
28 eval ($weps = 80.0)
29
30 ! ===== Non-bonded model =====
31 ! notice NB parameters depend on the GB model
32
33 ! Dielectric constant
34 eval ($eps = 4.0)
35
36 ! Inhibit distance
37 eval ($inhibit = 0.0)
38
39 if ($GB = 0) then
40
41     ! Non-bonded cuton
42     eval ($ctonnb = 10.0)
43
44     ! Non-bonded cutoff
45     eval ($ctofnb = 12.0)
46
47     ! Non-bonded cutnb
48     eval ($cutnb = 14.0)
49
50     ! Non-bonded tolerance
51     eval ($toler = 0.25)
52
53 end if
54
55 if ($GB = 1) then
56
57     ! Non-bonded cuton
58     eval ($ctonnb = 979.0)
59
60     ! Non-bonded cutoff
61     eval ($ctofnb = 989.0)

```

```

62
63     ! Non-bonded cutnb
64     eval ($cutnb = 999.0)
65
66     ! Non-bonded tolerance
67     eval ($toler = 999.0)
68
69 end if
70
71 ! ===== Distance filters =====
72 ! First distance filter (CB-CB distance)
73 eval ($firstfilter = 30.0)
74
75 ! Second distance filter (minimum sidechain-sidechain distance) for interaction
76 eval ($secondfilter = 12.01)
77
78 ! Third distance filter (minimum sidechain-sidechain distance) for minimization
79 eval ($thirdfilter = 3.0)
80
81 ! Increased cutoff for second distance filter
82 eval ($increasedcutoff = 12.1)
83
84 ! GB exclusion cutoff
85 eval ($dcut = 3.0)
86
87 ! ===== Surface Area Term =====
88 ! Surface area term (zero - not included)
89 eval ($sa = 1)
90
91 ! Surface backbone cutoff
92 eval ($sabbcutoff = 14.0)
93
94 ! Surface IJ sidechain distance filter
95 eval ($saijfilter = 7.0)
96
97 ! Surface correction factor for buried residues
98 eval ($buriedfactor = 1.0)
99
100 ! Surface correction factor for exposed residues
101 eval ($exposedfactor = 1.0)
102
103 ! Surface burial fraction threshold
104 eval ($threshold = 0.3)
105
106 ! Surface calculations probe radius
107 eval ($rh2o = 1.5)
108
109 ! Surface calculations accuracy
110 eval ($accu = 0.005)
111

```



```

112
113 ! ===== Restraints =====
114 ! Dihedral restraints force constant
115 eval ($dihecons = 200.0)
116
117 ! Dihedral restraints angle range
118 eval ($diherange = 5.0)
119
120 ! Dihedral restraints maximum number of assignments
121 eval ($nassign = 300)
122
123 ! Dihedral restraints scale
124 eval ($scale = 1.0)
125
126
127 ! ===== Minimization =====
128 ! Minimization number of steps for matrix I
129 eval ($nstepi = 15)
130
131 ! Minimization number of steps for matrix IJ
132 eval ($nstepij = 0)
133
134 ! Minimization number of steps for reconstruction
135 eval ($nstepReconstr = 0)
136
137 ! Minimization expected initial drop
138 eval ($drop = 10)
139
140 ! Minimization print frequency
141 eval ($nprint = 5)
142
143 ! Native rotamers
144 eval ($nativerot = 0)
145
146 ! ===== Reference energy =====
147 ! CASA model
148 eval ($casa = 0)
149
150 ! GB model, eps = 4 + SA
151 eval ($gbe4sa = 1)
152
153 ! ===== Output =====
154 ! Enriched output format
155 eval ($enrichedoutput = 1)
156
157 ! Decompose surface terms by type (n,p,a,i)
158 eval ($decompsurf = 0)
159
160 ! Debug
161 eval ($debug = 0)

```

7.2 inp/matrixI.inp

matrixI.inp

```
1  remarks
2  remarks  Compute energy matrix for protein design
3  remarks
4  remarks  by Anne Lopes, Marcel Schmidt-am-Busch, Thomas Gaillard and Thomas Simonson
5  remarks  Ecole Polytechnique, 2005-2011
6  remarks
7  remarks  matrix loop I precalculations
8  remarks  optional argument: positionI
9  remarks  this file: matrixI.inp
10 remarks
11
12
13 !=====
14 !
15 ! Should be run in the user's matrix directory
16 !
17 ! Updated by TS, SP, and KD, June 2013
18 ! - added support for ligands
19 ! - added support for acid/base activity
20 ! Updated by TG, January 2011:
21 ! - separate scripts for II loop and IJ loops
22 ! - inactive, active, and ligand positions in the same loop
23 ! - mutation space stream file
24 ! - native rotamer support
25 ! - GB ACE and HCT support
26 ! - ff99SB force field support
27 ! - changed rotamer library format (Pick, Rest, Nbrrot, Chis, Rota)
28 ! Updated by TG, November 2009:
29 ! - increased modularization (stream files)
30 ! - giant residues at all active positions
31 ! (instead of moving giant residues 999 and 998 around)
32 ! - improved the surface calculation decomposition
33 ! (3-body backbone-I-J effects were ignored)
34 ! Updated by TS and MSAB, January 2009
35 ! - Cbeta moved into sidechain
36 ! - took into account the desolvation of backbone by the sidechain of 999
37 ! - applied burial correction to the sidechain-sidechain ASA interaction
38 !=====
39
40
41 ! Set default matrix output file
42 eval ($matrixfile = "dat/matrix_I.dat")
43
44 ! Positions I to process
45 if ($exist_positionI = TRUE) then
```

```

46 eval ($firstI = decode($positionI))
47 eval ($lastI = decode($positionI))
48 eval ($matrixfile = "dat/matrix_I_" + $positionI + ".dat")
49 else
50 eval ($firstI = 1)
51 eval ($lastI = 9999)
52 end if
53
54 ! Parameters for this job
55 @MYLIB:parameters.str
56
57 !-----
58 ! Prepare system
59 !-----
60
61 ! Read topology and parameters
62 @CPD:lib/toppar.str
63
64 ! Read PSF
65 struct @setup.psf end
66
67 ! Read coordinates
68 coor @setup.pdb
69
70 ! Show unknown coordinates
71 write coor sele=(not known) end
72
73 ! Store initial coordinates to ref arrays
74 vector do (refx = x) (known)
75 vector do (refy = y) (known)
76 vector do (refz = z) (known)
77
78
79 ! Non-bonded energy options
80 @CPD:lib/nb.str
81
82 ! Read parameters for surface area energy term: store in fbeta
83 @MYLIB:phia.str
84
85 ! Read reference energies: store in harm
86 @MYLIB:refener.str
87
88 ! Read one letter codes for amino acids and any ligands: store in string1
89 @CPD:lib/oneletter.str
90 @MYLIB:oneletterLIGA.str
91
92 ! Define selections of interest
93 @MYLIB:sele.str
94 ! store1: inactive residues
95 ! store2: active residues

```

```

96 ! store3: ligand center atom
97 ! store5: ligand fitting atoms
98 ! store6: protein fitting atoms
99 ! store7: ligand backbone
100 ! store8: protein backbone
101 ! store9: whole backbone
102
103
104 if ($gb = 1) then
105     ! Read solvation radii for backbone
106     @CPD:lib/batom_read_bb.str
107     ! Save native b values in vz
108     vector do (vz = bsolv) (all)
109 end if
110
111 !-----
112 ! Loop l1 over residues at position I
113 !-----
114
115 ! Begin loop l1                ! modified to include ligand (Feb 2013)
116 for $i in id ( resid $firstI:$lastI and ( (name CA and (store1 or store2)) or store3 ) ) loop l1
117
118 ! Get current resid
119 vector show (resid) (id $i)
120 eval ($1 = decode($result))
121
122 ! Get current resname
123 vector show (resname) (id $i)
124 eval ($aa1 = $result)
125
126 ! Save current backbone resname
127 eval ($bbaa1 = $aa1)
128
129 ! Get current segid
130 vector show (segid) (id $i)
131 eval ($seg1 = $result)
132
133 ! Get current b-factor
134 vector show (b) (id $i)
135 eval ($b1 = $result)
136
137 ! Find type
138 if ($b1 = 1) then eval ($type1 = "inactive")
139 elseif ($b1 = 2) then eval ($type1 = "active")
140 end if
141 if ($seg1 = "LIGA") then eval ($type1 = "ligand") end if
142
143 ! Determines mutation space
144 eval ($mutation_space1 = "local/Mut/" + encode($1) + "_" + $type1 + "_" + $bbaa1 + ".dat")
145

```

```

146 ! Determines rotamer directories
147 if ($type1 = "ligand") then
148 eval ($rotlibdir1 = "LIGROTLIBDIR:")
149 if ($nativerot = 1) then
150 eval ($rotnatdir1 = "LIGROTNATDIR:")
151 end if
152 else
153 eval ($rotlibdir1 = "ROTLIBDIR:")
154 if ($nativerot = 1) then
155 eval ($rotnatdir1 = "ROTNATDIR:")
156 end if
157 end if
158
159
160 if ($sa = 1) then
161 ! Initialize storage vector: modified to include ligand (Feb 2013)
162 vector do (vx = 0.0) (all)
163 ! Surface of position I local backbone; save in vx
164 surf rh2o=$rh2o accu=$accu mode=access
165     sele((((name CA and not segid LIGA) or store3) and resid $1) around $sabbcutoff) and
166     store9 and not name H*) end
167 vector do (vx = rmsd) (((((name CA and not segid LIGA) or store3) and resid $1) around $sabbcutoff) and
168     store9 and not name H*)
169 end if
170
171 !-----
172 ! Loop laa1 over amino acid types for current residue I
173 !-----
174
175 ! Begin loop laa1
176 for $aa1 in ( @@$mutation_space1 ) loop laa1
177
178 ! Get number of library rotamers
179 eval ($nbrotlibfile1 = $rotlibdir1 + "Nbrot/" + $aa1 + ".dat")
180 @@$nbrotlibfile1
181 eval ($nbrotlib1 = $nbrot)
182 eval ($nbrot1 = $nbrotlib1)
183
184 ! Write number of rotamers
185 eval ($nbrotfile1 = "local/Nbrot/" + encode($1) + "_" + $aa1 + ".dat")
186 set display=$nbrotfile1 end
187 eval ($string = "$nbrot")
188 display eval ($string = $nbrot1)
189 close $nbrotfile1 end
190 set display=OUTPUT end
191 if ($nativerot = 1) then
192 if ($aa1 = $bbaa1) then
193 ! Handle residue I sidechain native rotamers
194 @CPD:lib/nativerotI.str
195 end if

```

```

196 end if
197
198 !-----
199 ! Loop lrot1 over rotamers for current residue I
200 !-----
201
202 ! Initialize rotamer counter for residue I
203 eval ($rot1 = 1)
204
205 ! Begin loop lrot1
206 while ($rot1 <= $nbrot1) loop lrot1
207
208 ! Place residue I sidechain rotamer
209 ! unless this is a native rotamer OR a ligand rotamer
210 eval ($doplaceI = 1)
211 if ($nativerot = 1) then
212 if ($aa1 = $bbaa1) then
213 if ($rot1 > $nbrotlib1) then
214 eval ($doplaceI = 0)
215 end if
216 end if
217 end if
218 if ($stype1 = "ligand") then ! read coordinates, do not fit
219 eval ($doplaceI = 0)
220 eval ($rot1file1 = $rotlibdir1 + "Rota/" + $aa1 + "_" + encode($rot1) + ".pdb")
221 coor sele=(segid LIGA and resnam $aa1) @$rot1file1
222 vector ident (store3) (segid LIGA and tag and known)
223 end if
224 if ($doplaceI = 1) then
225 @CPD:lib/placeI.str
226 end if
227
228
229
230 if ($gb = 1) then
231 ! Compute residue I sidechain solvation radii,
232 ! which will be used during minimization
233 @CPD:lib/batom_scI.str
234 end if
235
236 ! Minimize residue I sidechain rotamer
237 @CPD:lib/minI.str
238
239 if ($gb = 1) then
240 ! Update residue I sidechain solvation radii
241 @CPD:lib/batom_scI.str
242 end if
243
244 ! Compute residue I sidechain interactions
245 @CPD:lib/interactionsI.str

```

```

246
247 ! Write residue I sidechain rotamer coordinates
248 eval ($rot1file1 = "local/Rota/" + encode($1) + "_" + $aa1 + "_" + encode($rot1) + ".pdb")
249 REMARK position $1 $type1 $bbaa1 : rotamer $aa1 $rot1
250 if ($nativerot = 1) then
251   if ($aa1 = $bbaa1) then
252     if ($rot1 > $nbrotlib1) then
253       REMARK position $1 $type1 $bbaa1 : native rotamer $rot1
254     end if
255   end if
256 end if
257 write coor output=$rot1file1 sele=(resid $1 and resn $aa1 and not store9) end
258
259 ! Restore initial coordinates for sidechain I
260 coor init sele (resid $1 and resn $aa1 and not store9) end
261 vector do (x = refx) (resid $1 and resn $bbaa1 and not store9)
262 vector do (y = refy) (resid $1 and resn $bbaa1 and not store9)
263 vector do (z = refz) (resid $1 and resn $bbaa1 and not store9)
264
265 ! Increment rotamer counter for residue I
266 eval ($rot1 = $rot1 + 1)
267
268 ! End loop lrot1
269 end loop lrot1
270
271 ! Restore initial backbone resname for residue I
272 vector do (resname = $bbaa1) (resid $1 and store9)
273
274 ! End loop laa1
275 end loop laa1
276
277 ! End loop l1
278 end loop l1
279
280 ! Close matrix output file
281 close $matrixfile end
282
283 stop
284
285
286

```

7.3 inp/matrixIJ.inp

matrixIJ.inp

```
1  remarks
2  remarks  Compute energy matrix for protein design
3  remarks
4  remarks  by Anne Lopes, Marcel Schmidt-am-Busch, Thomas Gaillard and Thomas Simonson
5  remarks  Ecole Polytechnique, 2005-2011
6  remarks
7  remarks  matrix loop I / loop J calculations
8  remarks  optional arguments: positionI / positionJ
9  remarks  this file: matrixIJ.inp
10 remarks
11
12
13 !=====
14 !
15 ! Should be run in the user's matrix directory
16 !
17 ! Updated by TS, SP, and KD, June 2013
18 ! - added support for ligands
19 ! - added support for acid/base activity
20 ! Updated by TG, January 2011:
21 ! - separate scripts for II loop and IJ loops
22 ! - inactive, active, and ligand positions in the same loop
23 ! - mutation space stream file
24 ! - native rotamer support
25 ! - GB ACE and HCT support
26 ! - ff99SB force field support
27 ! - changed rotamer library format (Pick, Rest, Nbrrot, Chis, Rota)
28 ! Updated by TG, November 2009:
29 ! - increased modularization (stream files)
30 ! - giant residues at all active positions
31 ! (instead of moving giant residues 999 and 998 around)
32 ! - improved the surface calculation decomposition
33 ! (3-body backbone-I-J effects were ignored)
34 ! Updated by TS and MSAB, January 2009
35 ! - Cbeta moved into sidechain
36 ! - took into account the desolvation of backbone by the sidechain of 999
37 ! - applied burial correction to the sidechain-sidechain ASA interaction
38 !=====
39
40
41 ! Set default matrix output file
42 eval ($matrixfile = "dat/matrix_IJ.dat")
43
44 ! Positions I to process
45 if ($exist_positionI = TRUE) then
46 eval ($firstI = decode($positionI))
47 eval ($lastI = decode($positionI))
48 eval ($matrixfile = "dat/matrix_IJ_" + $positionI + ".dat")
```



```

49 else
50 eval ($firstI = 1)
51 eval ($lastI = 9999)
52 end if
53
54 ! Positions J to process
55 if ($exist_positionJ = TRUE) then
56 eval ($firstJ = decode($positionJ))
57 eval ($lastJ = decode($positionJ))
58 eval ($matrixfile = "dat/matrix_IJ_" + $positionI + "_" + $positionJ + ".dat")
59 else
60 eval ($firstJ = 1)
61 eval ($lastJ = 9999)
62 end if
63
64 ! Parameters for this job
65 @MYLIB:parameters.str
66
67 ! Debugging output
68 if ($debug = 1) then
69     set echo=true end
70 else
71     set echo=false end
72 end if
73
74 !-----
75 ! Prepare system
76 !-----
77
78 ! Read topology and parameters
79 @CPD:lib/toppar.str
80
81 ! Read PSF
82 struct @setup.psf end
83
84 ! Read coordinates
85 coor @setup.pdb
86
87 ! Show unknown coordinates
88 write coor sele=(not known) end
89
90 ! Store initial coordinates to ref arrays
91 vector do (refx = x) (known)
92 vector do (refy = y) (known)
93 vector do (refz = z) (known)
94
95 ! Non-bonded energy options
96 @CPD:lib/nb.str
97
98 ! Read parameters for surface area energy term: store in fbeta

```

```

99 @MYLIB:phia.str
100
101 ! Read reference energies: store in harm
102 @MYLIB:refener.str
103
104 ! Read one letter codes for amino acids and any ligands: store in string1
105 @CPD:lib/oneletter.str
106 @MYLIB:oneletterLIGA.str
107
108 ! Define selections of interest
109 @MYLIB:sele.str
110 ! store1: inactive residues
111 ! store2: active residues
112 ! store3: ligand center atom
113 ! store5: ligand fitting atoms
114 ! store6: protein fitting atoms
115 ! store7: ligand backbone
116 ! store8: protein backbone
117 ! store9: whole backbone
118
119
120 if ($gb = 1) then
121     ! Read solvation radii for backbone
122     @CPD:lib/batom_read_bb.str
123 end if
124
125 !-----
126 ! Loop l1 over residues at position I
127 !-----
128
129 ! Begin loop l1          ! Modified to include ligand (Feb 2013)
130 for $i in id (resid $firstI:$lastI and ((name CA and (store1 or store2)) or store3)) loop l1
131
132 ! Get current resid
133 vector show (resid) (id $i)
134 eval ($1 = decode($result))
135
136 ! Get current resname
137 vector show (resname) (id $i)
138 eval ($aa1 = $result)
139
140 ! Save current backbone resname
141 eval ($bbaa1 = $aa1)
142
143 ! Get current segid
144 vector show (segid) (id $i)
145 eval ($seg1 = $result)
146
147 ! Get current b-factor
148 vector show (b) (id $i)

```

```

149 eval ($b1 = $result)
150
151 ! Find type
152 if ($b1 = 1) then eval ($type1 = "inactive")
153 elseif ($b1 = 2) then eval ($type1 = "active")
154 end if
155 if ($seg1 = "LIGA") then eval ($type1 = "ligand") end if
156
157 ! Determines mutation space
158 eval ($mutation_space1 = "local/Mut/" + encode($1) + "_" + $type1 + "_" + $bbaa1 + ".dat")
159
160 ! Determines rotamer directories
161 if ($type1 = "ligand") then
162 eval ($rotlibdir1 = "LIGROTLIBDIR:")
163 if ($nativerot = 1) then
164 eval ($rotnatdir1 = "LIGROTNATDIR:")
165 end if
166 else
167 eval ($rotlibdir1 = "ROTLIBDIR:")
168 if ($nativerot = 1) then
169 eval ($rotnatdir1 = "ROTNATDIR:")
170 end if
171 end if
172
173 !! MOVED INSIDE ROT1 LOOP BECAUSE OF LACK OF VARIABLES
174 !if ($sa = 1) then
175 !! Initialize storage vector
176 !vector do (vx = 0.0) (all)
177 !! Surface of position I local backbone; save in vx
178 !! COULD BE READ FROM LOCAL STORAGE
179 !surf rh2o=$rh2o accu=$accu mode=access
180 !     sele=(((name CA and resid $1) around $sabbcutoff) and
181 !     store9 and not name H*) end
182 !vector do (vx = rmsd) (((name CA and resid $1) around $sabbcutoff) and
183 !     store9 and not name H*)
184 !end if
185
186 !-----
187 ! Loop laa1 over amino acid types for current residue I
188 !-----
189
190 ! Begin loop laa1
191 for $aa1 in ( @@ $mutation_space1 ) loop laa1
192
193 ! Get number of rotamers
194 eval ($nbrotfile1 = "local/Nbrot/" + encode($1) + "_" + $aa1 + ".dat")
195 @@ $nbrotfile1
196 eval ($nbrot1 = $nbrot)
197 if ($nativerot = 1) then
198 if ($aa1 = $bbaa1) then

```

```

199 eval ($nbrotlib1 = $nbrotlib)
200 eval ($nbrotnat1 = $nbrotnat)
201 end if
202 end if
203
204
205 !-----
206 ! Loop lrot1 over rotamers for current residue I
207 !-----
208
209 ! Determines rotamer space
210 eval ($rotamers1 = "local/EnrFltr/" + encode($1) + "_" + $aa1 + ".dat")
211
212 ! Begin loop lrot1
213 for $rot1 in ( @@$rotamers1 ) loop lrot1
214
215 ! Read residue I sidechain rotamer
216 @CPD:lib/readI.str
217
218 ! Recreate store3 if ligand coordinates change
219 vector ident (store3) (segid LIGA and tag and known)
220
221 if($sa = 1) then
222 ! INDEPENDENT OF AA1/ROT1 COULD BE PUT HIGHER IN THE SCRIPT
223 ! Initialize storage vector
224 vector do (vx = 0.0) (all)
225 ! Surface of position I local backbone; save in vx          ! Modified to include ligand (Feb 2013)
226 ! COULD BE READ FROM LOCAL STORAGE
227 surf rh2o=$rh2o accu=$accu mode=access
228     sele=((((name CA and not segid LIGA) or store3) and resid $1) around $sabbcutoff) and
229     store9 and not name H*) end
230 vector do (vx = rmsd) (((((name CA and not segid LIGA) or store3) and resid $1) around $sabbcutoff) and
231     store9 and not name H*)
232
233     ! Precompute surface areas
234     @CPD:lib/interactionsI_casa.str
235
236 end if
237
238
239 !-----
240 ! Loop l2 over residues at position J
241 !-----
242
243 ! Begin loop l2          ! Modified to include ligand (Feb 2013)
244 for $j in id (resid $firstJ:$lastJ and ((name CA and (store1 or store2)) or store3)) loop l2
245
246 ! Get current resid
247 vector show (resid) (id $j)
248 eval ($2 = decode($result))

```

```

249
250 ! Get current resname
251 vector show (resname) (id $j)
252 eval ($aa2 = $result)
253
254 ! Save current backbone resname
255 eval ($bbaa2 = $aa2)
256
257 ! Get current segid
258 vector show (segid) (id $j)
259 eval ($seg2 = $result)
260
261 ! Get current b-factor
262 vector show (b) (id $j)
263 eval ($b2 = $result)
264
265 ! Find type
266 if ($b2 = 1) then eval ($type2 = "inactive")
267 elseif ($b2 = 2) then eval ($type2 = "active")
268 end if
269 if ($seg2 = "LIGA") then eval ($type2 = "ligand") end if
270
271 ! Determines mutation space
272 eval ($mutation_space2 = "local/Mut/" + encode($2) + "_" + $type2 + "_" + $bbaa2 + ".dat")
273
274 ! Determines dihedral pick, restraints, and rotamers directories
275 if ($type2 = "ligand") then
276     eval ($rotlibdir2 = "LIGROTLIBDIR:")
277     eval ($rotnatdir2 = "LIGROTNATDIR:")
278 else
279     eval ($rotlibdir2 = "ROTLIBDIR:")
280     eval ($rotnatdir2 = "ROTNATDIR:")
281 end if
282
283 ! Initialize marker for a pair to be computed
284 eval ($mark = 1)
285
286 ! First distance filter
287 @CPD:lib/first_distance_filter.str
288
289 ! Test marker for a pair to be computed
290 if ($mark = 1) then
291
292     if ($sa = 1) then
293         ! Initialize storage vector
294         vector do (vx = 0.0) (all)
295         ! Surface of position J local backbone; save in vx          ! Modified to include ligand (Feb 2013)
296         ! COULD BE READ FROM LOCAL STORAGE
297         surf rh2o=$rh2o accu=$accu mode=access
298         sele=((((name CA and not segid LIGA) or store3) and resid $2)

```

```

299         around $sabbcutoff) and store9 and not name H*) end
300 vector do (vx = rmsd) (((((name CA and not segid LIGA) or store3) and resid $2)
301         around $sabbcutoff) and store9 and not name H*)
302 ! Initialize storage vector
303 vector do (vz = 0.0) (all)
304 ! Surface of positions I and J local backbone; save in vz          ! Modified to include ligand (Feb 2013)
305 surf rh2o=$rh2o accu=$accu mode=access
306     sele=((((name CA and not segid LIGA) or store3) and (resid $1 or resid $2))
307     around $sabbcutoff) and store9 and not name H*) end
308 vector do (vz = rmsd) (((((name CA and not segid LIGA) or store3) and (resid $1 or resid $2))
309     around $sabbcutoff) and store9 and not name H*)
310 end if
311
312
313 !-----
314 ! Loop laa2 over amino acid types for current residue J
315 !-----
316
317 ! Begin loop laa2
318 for $aa2 in ( @@$mutation_space2 ) loop laa2
319
320 ! Initialize previous rotamer number
321 eval ($rot2prev = -9999)
322
323 ! Get number of rotamers
324 eval ($nbrotfile2 = "local/Nbrot/" + encode($2) + "_" + $aa2 + ".dat")
325 @@$nbrotfile2
326 eval ($nbrot2 = $nbrot)
327 if ($nativerot = 1) then
328 if ($aa2 = $bbaa2) then
329 eval ($nbrotlib2 = $nbrotlib)
330 eval ($nbrotnat2 = $nbrotnat)
331 end if
332 end if
333
334 !-----
335 ! Loop lrot2 over rotamers for current residue J
336 !-----
337
338 ! Determines rotamer space
339 eval ($rotamers2 = "local/EnrFltr/" + encode($2) + "_" + $aa2 + ".dat")
340
341 ! Begin loop lrot2
342 for $rot2 in ( @@$rotamers2 ) loop lrot2
343
344 ! Read residue J sidechain rotamer
345 @CPD:lib/readJ.str
346
347 ! Recreate store3 if ligand coordinates change
348 vector ident (store3) (segid LIGA and tag and known)

```

```

349
350 ! Second distance filter
351 @CPD:lib/second_distance_filter.str
352
353 ! Test minimum distance between sidechains I and J
354 if ($dmin < $secondfilter) then
355     if ($dmin < $thirdfilter) then
356         ! Minimize residue I-J sidechain rotamers
357         @CPD:lib/minIJ.str
358     end if {end minimization}
359
360     ! Compute residue I-J sidechain interactions
361     @CPD:lib/interactionsIJ.str
362
363 end if {end second filter}
364
365 ! Restore sidechain I coordinates prior to IJ minimization
366 coor swap sele (resid $1 and resn $aa1 and not store9) end
367
368 ! Restore initial coordinates for sidechain J
369 coor init sele (resid $2 and resn $aa2 and not store9) end
370 vector do (x = refx) (resid $2 and resn $bbaa2 and not store9)
371 vector do (y = refy) (resid $2 and resn $bbaa2 and not store9)
372 vector do (z = refz) (resid $2 and resn $bbaa2 and not store9)
373
374 ! Increment rotamer counter for residue J
375 !eval ($rot2 = $rot2 + 1)
376
377 ! End loop lrot2
378 end loop lrot2
379
380 ! Restore initial backbone resname for residue J
381 vector do (resname = $bbaa2) (resid $2 and store9)
382
383 ! End loop laa2
384 end loop laa2
385
386 ! End test marker for a pair to be computed
387 end if
388
389 ! End loop l2
390 end loop l2
391
392 ! Restore initial coordinates for sidechain I
393 coor init sele (resid $1 and resn $aa1 and not store9) end
394 vector do (x = refx) (resid $1 and resn $bbaa1 and not store9)
395 vector do (y = refy) (resid $1 and resn $bbaa1 and not store9)
396 vector do (z = refz) (resid $1 and resn $bbaa1 and not store9)
397
398 ! Increment rotamer counter for residue I

```

```
399 ! eval ($rot1 = $rot1 + 1)
400
401 ! End loop lrot1
402 end loop lrot1
403
404 ! Restore initial backbone resname for residue I
405 vector do (resname = $bbaa1) (resid $1 and store9)
406
407 ! End loop laa1
408 end loop laa1
409
410 ! End loop l1
411 end loop l1
412
413 ! Close matrix output file
414 close $matrixfile end
415
416 stop
417
418
419
```

8 Implementation of Generalized Born solvent models in XPLOR

8.1 Introduction

The Generalized Born (GB) model [9–12] is an efficient and accurate implicit solvent model for biomolecular simulations and structure refinement. It describes the solvent around the biomolecule as a dielectric continuum. But the numerical complexities of an inhomogeneous solute/solvent dielectric system are effectively swept away and replaced by approximate, efficient, analytical formulas. The model thus allows one to compute the electrostatic interactions between a macromolecule and its surrounding solvent without explicitly including individual solvent molecules in the calculation. It can be used either to determine the energy of a single structure or to generate multiple structures by molecular dynamics or simulated annealing. Several recent review articles describe the theoretical background, the performance, and the ongoing progress of the GB model; see eg [13–16]. Two GB variants have been implemented in X-PLOR [1] and CNS [5]. The first is termed GB/ACE (Schaefer & Karplus, *J. Phys. Chem.*, 1996, 100:1578), for ‘Analytical Continuum Electrostatics’; the second is termed GB/HCT, for ‘Hawkins, Cramer & Truhlar’ (HCT, *Chem. Phys. Lett.*, 1995, 246:122). We emphasize at the outset that the GB solvation model describes the solvent response to the charges and Coulomb potential of the solute. Therefore, it is meaningless to use GB in a simulation or structure refinement where the ordinary electrostatics energy term is turned off.

The Theory section below reviews the GB/ACE and GB/HCT models. Expressions of the solvation energies and forces are given. This section can be skipped by those already familiar with the model. The following section, Syntax, gives the necessary syntax and the default options for using GB in XPLOR. The last section, Installation and Testing, describes the source file organization, the method to merge the GB source code with an existing XPLOR distribution, and the execution of test files.

8.2 Theory

8.2.1 GB energy

In the world of continuum electrostatics, a biomolecular solute is viewed as a set of (fractional) atomic charges in a cavity delimited by the solute surface, embedded in a high dielectric solvent medium [17]. The electrostatic energy E^{elec} is the sum of the Coulomb interaction energies between all solute charges and a solvation term ΔE^{solv} ; the latter includes the interaction energies of each solute charge with solvent (its “self-energy”), and a solvent-screening contribution to the interaction energies between solute charges:

$$E^{\text{elec}} = \sum_{i < j} \frac{q_i q_j}{r_{ij}} + \Delta E^{\text{solv}} \quad (1)$$

$$\Delta E^{\text{solv}} = \sum_i \Delta E_i^{\text{self}} + \sum_{i < j} \Delta E_{ij}^{\text{int}}. \quad (2)$$

In the GB model, the solvent contribution $\Delta E_{ij}^{\text{int}}$ to the interaction energy between the charges q_i and q_j is approximated by [9]:

$$\Delta E_{ij}^{\text{int}} = -\frac{\tau q_i q_j}{(r_{ij}^2 + b_i b_j \exp[-r_{ij}^2/4b_i b_j])^{1/2}} \quad (3)$$

where r_{ij} is the distance between the charges, τ is given by

$$\tau = 1 - 1/\epsilon_w, \quad (4)$$

ϵ_w is the solvent dielectric constant, and b_i is the ‘solvation radius’ of charge i . By analogy to the case of a single charge in a spherical cavity, b_i is defined by

$$\Delta E_i^{\text{self}} = -\frac{\tau q_i^2}{2b_i}, \quad (5)$$

where ΔE_i^{self} is the self-energy of charge i . By partitioning the solute into atomic volumes (following Lee & Richards, for example [18]), one can express the self-energy ΔE_i^{self} as a sum over all the solute atoms [10, 11]:

$$\Delta E_i^{\text{self}} = -\frac{\tau q_i^2}{2R_i} + \tau q_i^2 \sum_{k \neq i} E_{ik}^{\text{self}}, \quad (6)$$

where R_i is a constant atomic radius to be determined (close to the van der Waals radius) and E_{ik}^{self} is related to the integral of the electrostatic energy over the volume of atom k . Notice that the charges of the other atoms, q_k , do not appear here. The effect of these atoms is merely to exclude solvent from the vicinity of atom i [19].

The volume integral E_{ik} is approximated in two steps. The first step is to approximate the electric field by the ‘Coulombic field’ of charge i [19]. This is simply the unscreened field that would exist if q_i were in a vacuum; it radiates uniformly in all directions and falls off as $1/r^2$ with distance; the corresponding energy density is $1/r^4$. The next step is to calculate the integral of $1/r^4$ over the volume of atom k . The different GB variants do this in different ways. In GB/ACE, for example, Schaefer & Karplus assume the density of each solute atom is a gaussian centered at the atom’s position. The integral E_{ik} then has a tractable form, which can be approximated by interpolating between a Gaussian form at short ranges and a $1/r^4$ form at long range, leading to the Ansatz [11]:

$$E_{ik}^{\text{self}} = \frac{1}{\omega_{ik}} \exp(-r_{ik}^2/\sigma_{ik}^2) + \frac{V_k}{8\pi} \left(\frac{r_{ik}^3}{r_{ik}^4 + \mu_{ik}^4} \right)^4. \quad (7)$$

Here, ω_{ik} and μ_{ik} are simple functions of the atomic volume V_k , the atomic radii R_i , R_k ($= [3V_k/4\pi]^{1/3}$), and an adjustable ‘smoothing’ parameter α which determines the width of the atomic gaussian distributions (see below). The atomic charges are taken directly from the existing force field. The adjustable parameters of the model are then the volumes V_k and the smoothing parameter α . Ionic strength is not included, although methods to do so have been proposed [20, 21]. Volumes V_k can be either calculated using Voronoi polyhedra (using an external program [18] and reading them into X-PLOR), or assigned values from existing

libraries [11, 20, 22]. Note that the V_k are considered to be constants, independent of the solute conformation. This is essential to obtain tractable expressions for the GB forces (see below).

With the above self-energy approximations, ΔE_i^{self} can sometimes become positive, so that the (necessarily positive) solvation radius can no longer be defined by Eq. (5). Therefore, we use a definition proposed by Schaefer et al. [23]:

$$\begin{aligned} b_i &= -\frac{\tau q_i^2}{2\Delta E_i^{\text{self}}} && \text{if } \Delta E_i^{\text{self}} \leq E_{\text{min}} = -\frac{\tau q_i^2}{2b_{\text{max}}} \\ &= b_{\text{max}} \left(2 - \frac{\Delta E_i^{\text{self}}}{E_{\text{min}}} \right) && \text{if } \Delta E_i^{\text{self}} \geq E_{\text{min}} \end{aligned} \quad (8)$$

Here, b_{max} is an upper limit for the solvation radius, which can be set to the largest linear dimension of the solute, for example. This definition leads to continuous energies and forces.

8.2.2 Calculation of forces

Interaction energy term We first consider the GB ‘interaction’ term, on the far right of Eq. (2), and its gradient ∇_n with respect to the position of solute particle n . Noting that the solvation radii b_i, b_j depend on all the atomic positions and using the chain rule for differentiation, we have:

$$\nabla_n \sum_{i<j} \Delta E_{ij}^{\text{int}} = \sum_{i<j} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial r_{ij}} \nabla_n r_{ij} + \sum_{i<j} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_i} \nabla_n b_i + \sum_{i<j} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_j} \nabla_n b_j \quad (9)$$

Only terms with $i = n$ or $j = n$ contribute to the first sum on the right. The second sum can be written

$$\sum_{i<j} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_i} \nabla_n b_i = \frac{1}{2} \sum_i \left(\sum_{j \neq i} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_i} \right) \frac{\partial b_i}{\partial \Delta E_i^{\text{self}}} \nabla_n \Delta E_i^{\text{self}} \quad (10)$$

The quantity in parentheses will be denoted $dE_i^{\text{int,b}}$, since, for a given conformation, it depends only on i . The last quantity on the right can be written:

$$\begin{aligned} \nabla_n \Delta E_i^{\text{self}} &= \sum_{k \neq i} \nabla_n E_{ik}^{\text{self}} = \nabla_n E_{in}^{\text{self}} && \text{if } i \neq n \\ &= \sum_{k \neq n} \nabla_n E_{nk}^{\text{self}} && \text{if } i = n. \end{aligned} \quad (11)$$

Grouping the second and third terms on the right of (9) and rearranging the first, we obtain:

$$\nabla_n \sum_{i<j} \Delta E_{ij}^{\text{int}} = \sum_{i \neq n} \left(\frac{\partial \Delta E_{in}^{\text{int}}}{\partial r_{in}} + dE_n^{\text{int,b}} \frac{\partial b_n}{\partial \Delta E_n^{\text{self}}} \frac{\partial E_{ni}^{\text{self}}}{\partial r_{in}} + dE_i^{\text{int,b}} \frac{\partial b_i}{\partial \Delta E_i^{\text{self}}} \frac{\partial E_{in}^{\text{self}}}{\partial r_{in}} \right) \frac{r_n - r_i}{r_{in}} \quad (12)$$

with

$$dE_i^{\text{int,b}} = \sum_{j \neq i} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_i} \quad (13)$$

$$\begin{aligned} \frac{\partial b_n}{\partial \Delta E_n^{\text{self}}} &= -\frac{b_n}{\Delta E_n^{\text{self}}} && \text{if } \Delta E_n^{\text{self}} \leq E_{\text{min}} = -\frac{\tau q_n^2}{2b_{\text{max}}} \\ &= -\frac{b_{\text{max}}}{E_{\text{min}}} && \text{if } \Delta E_n^{\text{self}} \geq E_{\text{min}} \end{aligned} \quad (14)$$

The quantities b_i and $dE_i^{\text{int,b}}$ can be ‘precalculated’, so that obtaining the force on atom n requires only a loop over all solute atoms. In (12), the derivatives of $\Delta E_{in}^{\text{int}}$ are the same for GB/ACE and GB/HCT:

$$\frac{1}{r_{in}} \frac{\partial \Delta E_{in}^{\text{int}}}{\partial r_{in}} = \frac{\tau q_i q_j}{\left[r_{ij}^2 + b_i b_j \exp\left(-\frac{r_{ij}^2}{4b_i b_j}\right) \right]^{3/2}} \left(1 - \frac{1}{4} \exp\left(-\frac{r_{ij}^2}{4b_i b_j}\right) \right) \quad (15)$$

$$dE_i^{\text{int,b}} = \sum_{j \neq i} \frac{\frac{1}{2} \tau q_i q_j b_j \exp\left(-\frac{r_{ij}^2}{4b_i b_j}\right)}{\left[r_{ij}^2 + b_i b_j \exp\left(-\frac{r_{ij}^2}{4b_i b_j}\right) \right]^{3/2}} \left(1 + \frac{r_{ij}^2}{4b_i b_j} \right) \quad (16)$$

GB/ACE self-energy term The self-energy and the associated forces depend on the GB variant. With GB/ACE,

$$\frac{1}{r_{ij}} \frac{\partial E_{ij}^{\text{self}}}{\partial r_{ij}} = -\frac{2}{\omega_{ij} \sigma_{ij}^2} \exp\left(-\frac{r_{ik}^2}{\sigma_{ik}^2}\right) + \frac{V_j}{2\pi} \left(\frac{r_{ij}^{10}}{r_{ij}^4 + \mu_{ij}^4} \right)^5 \left(3(r_{ij}^4 + \mu_{ij}^4) - 4r_{ij}^4 \right). \quad (17)$$

The parameters ω_{ij} , σ_{ij} , μ_{ij} are defined by:

$$\frac{1}{\omega_{ik}} = \frac{4}{3\pi \alpha_{ik}^3} (Q_{ik} - \arctan Q_{ik}) \frac{1}{\alpha_{ik} R_k} \quad (18)$$

$$\sigma_{ik}^2 = \frac{3(Q_{ik} - \arctan Q_{ik})}{(3 + f_{ik}) Q_{ik} - 4 \arctan Q_{ik}} \alpha_{ik}^2 R_{ik}^2 \quad (19)$$

$$Q_{ik} = \frac{q_{ik}^2}{(2q_{ik}^2 + 1)^{1/2}} \quad (20)$$

$$f_{ik} = \frac{2}{q_{ik}^2 + 1} - \frac{1}{2q_{ik}^2 + 1} \quad (21)$$

$$q_{ik}^2 = \frac{\pi}{2} \left(\frac{\alpha_{ik} R_k}{R_i} \right)^2 \quad (22)$$

$$\alpha_{ik} = \text{Max}(\alpha, R_i/R_k) \quad (23)$$

$$\mu_{ik} = \frac{77\pi\sqrt{2}R_i}{512(1 - 2\pi^{3/2}\sigma_{ik}^3) \frac{R_i}{\omega_{ik} V_k}} \quad (24)$$

$$V_k = \frac{4}{3} \pi R_k^3 \quad (25)$$

GB/HCT self-energy With GB/HCT, the self-energy contribution E_{ik}^{self} is given by [10]

$$4E_{ik}^{\text{self}} = \frac{1}{L_{ik}} - \frac{1}{U_{ik}} + \frac{r_{ik}}{4} \left(\frac{1}{U_{ik}^2} - \frac{1}{L_{ik}^2} \right) + \frac{1}{2r_{ik}} \ln \frac{L_{ik}}{U_{ik}} + \frac{R_k^2}{4r_{ik}} \left(\frac{1}{L_{ik}^2} - \frac{1}{U_{ik}^2} \right), \quad (26)$$

where

$$\begin{aligned} L_{ik} &= 1 & \text{if } r_{ik} + R_k &\leq R_i, \\ L_{ik} &= R_i & \text{if } r_{ik} - R_k &\leq R_k < r_{ik} + R_k, \\ L_{ik} &= r_{ik} - R_k & \text{if } R_i &\leq R_k < r_{ik} - R_k, \end{aligned} \quad (27)$$

$$\begin{aligned} U_{ik} &= 1 & \text{if } r_{ik} + R_k &\leq R_i, \\ U_{ik} &= r_{ik} - R_k & \text{if } R_i &< r_{ik} + R_k. \end{aligned} \quad (28)$$

The corresponding gradient is given by:

$$\begin{aligned} \frac{4}{r_{ik}} \frac{\partial E_{ik}^{\text{self}}}{\partial r_{ik}} &= -\frac{1}{r_{ik}} \left(\frac{L'_{ik}}{L_{ik}^2} - \frac{U'_{ik}}{U_{ik}^2} \right) + \frac{1}{4r_{ik}} \left(\frac{1}{U_{ik}^2} - \frac{1}{L_{ik}^2} \right) - \frac{1}{2} \left(\frac{U'_{ik}}{U_{ik}^3} - \frac{L'_{ik}}{L_{ik}^3} \right) \\ &- \frac{1}{2r_{ik}^3} \ln \frac{L_{ik}}{U_{ik}} + \frac{1}{2r_{ik}^2} \left(\frac{L'_{ik}}{L_{ik}} - \frac{U'_{ik}}{U_{ik}} \right) - \frac{R_k^2}{4r_{ik}^3} \left(\frac{1}{L_{ik}^2} - \frac{1}{U_{ik}^2} \right) - \frac{R_k^2}{2r_{ik}^2} \left(\frac{L'_{ik}}{L_{ik}^3} - \frac{U'_{ik}}{U_{ik}^3} \right) \end{aligned} \quad (29)$$

with $L'_{ik} = \partial L_{ik} / \partial r_{ik}$, $U'_{ik} = \partial U_{ik} / \partial r_{ik}$. The radii R_k are calculated from the atomic volumes as in Eq. (25), then reduced by a scaling factor $S_k \leq 1$ which depends only on the chemical type of atom k . Reasonable values are given in Table 1 of [10].

This basic model was modified by Onufriev et al [20] to improve performance for proteins. The self-energy in Eq. (6) is replaced by:

$$\Delta E_i^{\text{self}} = -\frac{\tau q_i^2}{2b_i} \quad (30)$$

$$b_i = \left[(R_i - \rho_0)^{-1} - \lambda \sum_{k \neq i} E_{ik}^{\text{self}} \right]^{-1} - \delta \quad (31)$$

In other words, the atomic radius R_i is reduced by a constant offset ρ_0 , the self-energy contribution E_{ik}^{self} is scaled by a constant factor λ , and the solvation radius b_i is reduced by a constant offset δ . The values $\lambda = 1.4$, $\rho_0 = 0.09 \text{ \AA}$ and $\delta = 0.15 \text{ \AA}$ were used in [20].

8.2.3 Pairs of interacting groups

In structure refinement, it is often necessary to use a model in which different parts of the macromolecule are artificially duplicated, for example a protein side chain that is disordered and occupies multiple positions in a crystal structure. To allow for these situations, both X-PLOR [1] and CNS [5] view the system formally as a set of ‘‘pairs of interacting groups’’. Usually, there is only one such pair: the macromolecule interacting with itself:

$$M \leftrightarrow M,$$

where M is the macromolecule and \leftrightarrow indicates an interaction. In the case of a single disordered protein side chain thought to have two main conformations, one would normally consider a protein P with two copies of the side chain: S_1 and S_2 , leading to the following pairs of interacting groups:

$$\begin{aligned} P \setminus \{S_1, S_2\} &\leftrightarrow P \setminus \{S_1, S_2\} \\ P \setminus \{S_1, S_2\} &\leftrightarrow S_1; \text{ weight of } 1/2 \\ P \setminus \{S_1, S_2\} &\leftrightarrow S_2; \text{ weight of } 1/2, \end{aligned}$$

where $P \setminus \{S_1, S_2\}$ represents the protein without the disordered side chain and the protein– S interactions are weighted by $1/2$ because there are two copies of S . The two copies of S do not interact with each other. This formalism is implemented in X-PLOR through the **constraints interaction** statement (for an example, see the gbttests/testfirst.inp test case).

The same formalism applies to the GB energy terms. If the interacting groups are denoted A_p, B_p with $p = 1, N$, their pairs take the form $P_p = A_p \times B_p = \{(i, j); i \in A_p, j \in B_p\}$. There

are N pairs of groups P_p and each has a weight w_p . The GB interaction and self energies take the form:

$$\Delta E^{\text{int}} = \frac{1}{2} \sum_{p=1}^N w_p \left(\sum_{i \in A_p, j \in B_p} \Delta E_{ij}^{\text{int}} \right) \quad (32)$$

$$\Delta E^{\text{self}} = \sum_{p=1}^N w_p \sum_{i \in A_p, j \in B_p} \left(-\frac{\tau q_i^2}{2R_i} \delta_{ij} + \tau q_i^2 E_{ij}^{\text{self}} \right) \quad (33)$$

These equations generalize Eqs. (3), (6), which correspond to a single pair $P_1 = M \times M$, M being the whole macromolecule. The factor $\frac{1}{2}$ in Eq. (32) corrects for double counting of i, j and j, i terms; δ_{ij} is the Kronecker symbol.

8.2.4 Crystal symmetry

The GB model has been implemented for systems with symmetry (crystallographic or otherwise); for details, see Moulinier et al, 2003 [7].

8.3 Syntax

8.3.1 GB energy terms

The GB solvation energy is divided into a self-energy term and an interaction energy term, corresponding to the two terms on the right of (Eq. 2):

$$E_{GBSOLV} = E_{GBSELF} + E_{GBINT}$$

They are available to the user through the variables \$GBSE and \$GBIN. They are activated by the **flags** statement in the usual way:

```
flags include gbse gbin end
```

They are inactive by default.

8.3.2 Setting the GB options

All the parameters of the GB solvent model are under user control, with sensible defaults. The setup of the atomic volumes is described further on. The other GB parameters are set up with the **nbonds** subcommand:

NBONDS <nbonds-statement> | <gborn-nbonds-statement> **END** applies to electrostatic, van der Waals, and GB energy terms.

<gborn-nbonds-statement> ::=

GBACE | **GBHCT** Exclusive flags activating the GB/ACE or the GB/HCT model.
Default: inactive.

WEPS=<real> Solvent dielectric constant. Default: 1 if GB is inactive, 80 if GB is active.

SMOOTH=<real> Determines the atomic widths in GB/ACE; denoted α in Eq. (23).

Default: 1.

LAMBDA=<real> Scaling factor for solvation radii in GB/HCT; denoted λ in Eq. (31).

Default: 1.

OFFSET=<real> Offset for atomic radii in GB/HCT; denoted ρ_0 in Eq. (31). Default:

0.

8.3.3 Setting up atomic volumes for GB

Two approaches can be used:

Volume libraries Two sets of ‘standard’ atomic volumes are available for proteins, in two force field parameter files: **param19.gb.pro** and **parameter.gb.inp**, located in \$GBX-PLOR/gbtoppar (see Fie Orgaization, below). These volumes are automatically read along with the other force field parameters. The first set was developed by Schaefer and coworkers [22] and modified and tested for protein simulations by Calimet et al [24], and is meant to be used with the Charmm19 topology (toph19.pro) and parameter set. The second was developed and tested by Onufriev et al [20] and is meant to be used with the Amber all-atom force field [25]. Other volume libraries are available in the literature and can be formatted for X-PLOR, for example nucleic acid libraries [26].

The syntax of the **NONBonded** subcommand is modified accordingly:

NONB <type> <real> <real> <real> <real> [<real> <real>] reads the Lennard-Jones parameters for a specified chemical type, as before; the first pair of reals is ϵ , σ ; the second pair is ϵ , σ for 1–4 non-bonded interactions. The last two reals are V , the atomic volume (Eqs. 6, 25), and S , the scaling parameter used for the HCT solvation radius (see text following Eq. 29). If the last two reals are omitted, V and S will both be set to 9999. Thus, for applications not using GB, there is backward compatibility with X-PLOR parameter files not set up for GB. But for applications using GB, V must be included in the parameter file for both GB/ACE and GB/HCT, and S must be included for GB/HCT.

Volumes calculated with an external program In some cases, it may be desirable to calculate the atomic volumes corresponding to a particular family of conformations and/or proteins, instead of relying on ‘standard’ values [27]. The standard GB/ACE volumes were obtained from atomic Voronoi volumes calculated for a large set of protein structures, then averaged over each chemical type [22], then reduced by a factor of 0.9 to account for systematic errors in the GB/ACE self-energy approximation [24]. Several programs have the capability to calculate Voronoi volumes for each individual atom of a particular protein (eg the VORONOI package of Fred Richards). If these are then stored in a particular field of a PDB coordinate file (for example the field normally used for the temperature factors, WMAIN), this information can be read into X-PLOR using the **coordinate** statement, then made available to the GB routines internally. To do this, the volumes must be copied into the RMSD array, then averaged over each chemical type using the **parameter reduce** statement:

```

1  coor @volumes.pdb          {read coordinate file with      }
2                               {atomic volumes in wmain field  }
3
4  vector do (rmsd = wmain) (all) {copy into rmsd field          }
5
6  flags exclude * include gbse gbin end {activate GB energy terms,      }
7                                         {so GB parameters will be reduced}
8
9  parameter reduce selection=(all) {average volumes over          }
10     overwrite=true mode=average end {each chemical type           }
11  end
12  flags include bonds angl dihe impr vdw elec {reactivate the other terms}

```

The atomic volumes, suitably averaged, are then available for GB calculations.

8.3.4 Examples

```

1  coordinates @protein.pdb
2
3  parameter
4  nbonds
5  tolerance=0.25 atom cdie trunc
6  nbxmod=5 vswitch e14fac=1. cutnb=15. ctonnb=13. ctofnb=14.
7
8  EPS=1. WEPS=80. smooth=1.3 gbace {GB options}
9
10 end
11 end
12
13 flags include gbse gbin end
14
15 minimize powell nstep=50 end

```

Minimization with GB/ACE

8.3.5 Molecular dynamics with GB/HCT

```

1  remarks  Asparagine MD with GB/HCT
2  remarks  this file: dyna.inp
3
4  topology
5  @GBXPLOr:gbttoppar/amber/topamber.inp {Amber topology file      }
6  @GBXPLOr:gbttoppar/amber/patches.pro {N- and C-terminal patches }
7  end {for Amber force field }
8
9  parameter
10 @GBXPLOr:gbttoppar/amber/paramber.gb.inp {Amber parameter file     }
11 end {including GB parameters }
12
13 segment
14 name="ASN1"
15 molecule name=ASN number=1 end
16 end
17 patch NASN refe=nil=(resid 1) end
18 patch CASN refe=nil=(resid 1) end
19
20 parameter
21 nbonds
22     atom cdie trunc
23     e14fac=0.8333333          ! use this to reproduce amber elec
24     cutnb 500. ctonnb 480. ctofnb 490. ! essentially no cutoff
25     tolerance=100.          ! only build the nonbonded list once
26     nbxmod 5 vswitch
27     wmin=1.0
28 end
29 end
30
31 parameters nbonds
32     EPS=1. WEPS=80. GBHCT          ! GB parameters
33     offset=0.09 lambda=1.33        ! GB parameters
34 end end
35
36 coor @volumes.pdb          ! PDB with volumes in wmain
37 vector do (RMSD = wmain) (all)    ! copy into rmsd
38 vector do (rmsd = rmsd * 0.9) (all) ! reduce volumes by 10%
39 flags include gbse gbin end

```



```
40 parameter reduce selection=(all) overwrite=true mode=average end end
41
42 coor @asn.pdb
43
44 ! Now run constant energy dynamics; random initial velocities
45
46 vector do (vx = maxwell(250)) (all)
47 vector do (vy = maxwell(250)) (all)
48 vector do (vz = maxwell(250)) (all)
49
50 dynamics verlet
51     nstep=500000 timest=0.001 {ps}      ! 500 ps dynamics
52     iasvel=current                       ! current velocities
53     nprint=250 iprfrq=250              ! statistics output
54 end
55
56 stop
```

References

- [1] BRÜNGER, A. T. *X-PLOR version 3.1, A System for X-ray crystallography and NMR*. Yale University Press, New Haven, 1992.
- [2] BRÜNGER, A. T., ADAMS, P. D., DELANO, W. L., GROS, P., GROSSE-KUNSTLEVE, R. W., JIANG, J., PANNU, N. S., READ, R. J., RICE, L. M., AND SIMONSON, T. The structure determination language of the Crystallography and NMR System. In *International Tables for Crystallography, Volume F*, M. Rossmann and E. Arnold, Eds. Dordrecht: Kluwer Academic Publishers, the Netherlands, 2001, pp. 710–720.
- [3] DAHIYAT, B. I., AND MAYO, S. L. De novo protein design: fully automated sequence selection. *Science* 278 (1997), 82–87.
- [4] SIMONSON, T., GAILLARD, T., MIGNON, D., SCHMIDT AM BUSCH, M., LOPES, A., AMARA, N., POLYDORIDES, S., SEDANO, A., DRUART, K., AND ARCHONTIS, G. Computational protein design: the Proteus software and selected applications. *J. Comput. Chem.* 0000 (2013), in press.
- [5] BRÜNGER, A., ADAMS, P., CLORE, G., DELANO, W., GROS, P., GROSSE-KUNSTLEVE, R., JIANG, J., KUSZEWSKI, J., NILGES, M., PANNU, N., READ, R., RICE, L., SIMONSON, T., AND WARREN, G. Crystallography and NMR System: a new software suite for macromolecular structure determination. *Acta Cryst. D54* (1998), 905–921.
- [6] BROOKS, B., BROOKS III, C. L., MACKERELL JR., A. D., NILSSON, L., PETRELLA, R. J., ROUX, B., WON, Y., ARCHONTIS, G., BARTELS, C., BORESCH, S., CAFLISCH, A., CAVES, L., CUI, Q., DINNER, A. R., FEIG, M., FISCHER, S., GAO, J., HODOSCEK, M., IM, W., KUCZERA, K., LAZARIDIS, T., MA, J., OVCHINNIKOV, V., PACI, E., PASTOR, R. W., POST, C. B., PU, J. Z., SCHAEFER, M., TIDOR, B., VENABLE, R. M., WOODCOCK, H. L., WU, X., YANG, W., YORK, D. M., AND KARPLUS, M. CHARMM: The biomolecular simulation program. *J. Comp. Chem.* 30 (2009), 1545–1614.
- [7] MOULINIER, L., CASE, D. A., AND SIMONSON, T. Xray structure refinement of proteins with the generalized Born solvent model. *Acta Cryst. D 59* (2003), 2094–2103.
- [8] LOPES, A., ALEKSANDROV, A., BATHELT, C., ARCHONTIS, G., AND SIMONSON, T. Computational sidechain placement and protein mutagenesis with implicit solvent models. *Proteins* 67 (2007), 853–867.
- [9] STILL, W. C., TEMPCZYK, A., HAWLEY, R., AND HENDRICKSON, T. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.* 112 (1990), 6127–6129.
- [10] HAWKINS, G. D., CRAMER, C., AND TRUHLAR, D. Pairwise descreening of solute charges from a dielectric medium. *Chem. Phys. Lett.* 246 (1995), 122–129.
- [11] SCHAEFER, M., AND KARPLUS, M. A comprehensive analytical treatment of continuum electrostatics. *J. Phys. Chem.* 100 (1996), 1578–1599.
- [12] QIU, D., SHENKIN, P., HOLLINGER, F., AND STILL, W. The GB/SA continuum model for solvation. A fast analytical method for the calculation of approximate Born radii. *J. Phys. Chem. A* 101 (1997), 3005–3014.

- [13] BASHFORD, D., AND CASE, D. Generalized Born models of macromolecular solvation effects. *Ann. Rev. Phys. Chem.* 51 (2000), 129–152.
- [14] ROUX, B., AND SIMONSON, T. Implicit solvent models. *Biophys. Chem.* 78 (1999), 1–20.
- [15] CRAMER, C., AND TRUHLAR, D. Implicit solvent models: equilibria, structure, spectra, and dynamics. *Chem. Rev.* 99 (1999), 2161–2200.
- [16] SIMONSON, T. Macromolecular electrostatics: continuum models and their growing pains. *Curr. Opin. Struct. Biol.* 11 (2001), 243–252.
- [17] KIRKWOOD, J., AND WESTHEIMER, F. The electrostatic influence of substituents on the dissociation constant of organic acids. *J. Chem. Phys.* 6 (1938), 506–512.
- [18] LEE, B., AND RICHARDS, F. The interpretation of protein structures: estimation of static accessibility. *J. Mol. Biol.* 55 (1971), 379–400.
- [19] SCHAEFER, M., AND FROEMMEL, C. A precise analytical method for calculating the electrostatic energy of macromolecules in aqueous solution. *J. Mol. Biol.* 216 (1990), 1045–1066.
- [20] ONUFRIEV, A., BASHFORD, D., AND CASE, D. A. Modification of the generalized Born model suitable for macromolecules. *J. Phys. Chem. B* 104 (2000), 3712–3720.
- [21] SRINIVASAN, J., TREVATAN, M., BEROZA, P., AND CASE, D. A. Application of a pairwise Generalized Born model to proteins and nucleic acids: inclusion of salt effects. *Theor. Chem. Acc.* 101 (1999), 426–434.
- [22] SCHAEFER, M., BARTELS, C., LECLERC, F., AND KARPLUS, M. Effective atom volumes for implicit solvent models: comparison between Voronoi volumes and minimum fluctuation volumes. *J. Comput. Chem.* 22 (2001), 1857–1879.
- [23] SCHAEFER, M., BARTELS, C., AND KARPLUS, M. Solution conformations and thermodynamics of structured peptides: molecular dynamics simulation with an implicit solvation model. *J. Mol. Biol.* 284 (1998), 835–847.
- [24] CALIMET, N., SCHAEFER, M., AND SIMONSON, T. Protein molecular dynamics with the Generalized Born/ACE solvent model. *Proteins* 45 (2001), 144–158.
- [25] CORNELL, W., CIEPLAK, P., BAYLY, C., GOULD, I., MERZ, K., FERGUSON, D., SPELLMEYER, D., FOX, T., CALDWELL, J., AND KOLLMAN, P. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.* 117 (1995), 5179–5197.
- [26] TSUI, V., AND CASE, D. A. Molecular dynamics simulations of nucleic acids with a Generalized Born model. *J. Am. Chem. Soc.* 122 (2000), 2489–2498.
- [27] WAGNER, F., AND SIMONSON, T. Implicit solvent models: combining an analytical formulation of continuum electrostatics with simple models of the hydrophobic effect. *J. Comput. Chem.* 20 (1999), 322–335.