The Proteus software for computational protein design

Thomas Simonson
Laboratoire de Biochimie, Ecole Polytechnique, Paris, France.
thomas.simonson@polytechnique.fr

Proteus is available free of charge to academic users under a Creative Commons BY-NC-SA license (version 4.0) from http://proteus.polytechnique.fr
Acknowledgements

The authors of the Proteus software are:

This manual is copyright Thomas Simonson and should be referenced as a publication.

Proteus is described in the following articles, which include theoretical and methodological developments:


In addition to the authors above, I am grateful to several colleagues for helpful discussions and/or contributions to Proteus development and/or to this documentation: David Allouche, Edouard Audit, Sophie Barbe, Christine Bathelt, Julien Bigot, Juan Cortes, Marie-Pierre Dreanic, Alfonso Jaramillo, Elena Michael, Thomas Schiex, Seydou Traoré. Part of Proteus was developed starting from the Xplor program by Axel T. Brünger. The protX section of this manual (part V) is adapted from the Xplor manual by Axel T. Brünger with his permission. Development of Proteus was supported by the Ecole Polytechnique, the Centre National de la Recherche Scien-
tifique, the Agence Nationale pour la Recherche, and the French supercomputing agency GENCI.

Thomas Simonson, Palaiseau, April 18, 2019
Contents

I Practical applications 13

1 Overview of programs and procedures 15
1.1 Directory structure and files . . . . . . . . . . . . . . . . . . . . . . . 15
  1.1.1 Proteus source directories . . . . . . . . . . . . . . . . . . . 15
  1.1.2 User directories for a Proteus application . . . . . . . . . . . . 16
1.2 Using Proteus for a typical protein system . . . . . . . . . . . . . . . 17
  1.2.1 System preparation for the matrix calculation . . . . . . . . . 17
  1.2.2 CPD setup: files to edit in $PROJ/lib . . . . . . . . . . . . . . 17
  1.2.3 The energy matrix . . . . . . . . . . . . . . . . . . . . . . . . 18
  1.2.4 Exploring sequence/rotamer space with protMC . . . . . . . . 21

2 Two test systems 23
2.1 The Syndecan-1 octapeptide . . . . . . . . . . . . . . . . . . . . . . . 23
  2.1.1 Common errors and suggestions for real projects . . . . . . . . 26
2.2 The Tiam1 PDZ domain . . . . . . . . . . . . . . . . . . . . . . . . . 27

3 Designing for binding with adaptive MC 29
3.1 Protocol to compute the matrix . . . . . . . . . . . . . . . . . . . . . 30
  3.1.1 System parameters and build . . . . . . . . . . . . . . . . . . . 30
  3.1.2 Matrix calculation . . . . . . . . . . . . . . . . . . . . . . . . . 31
  3.1.3 Computing the unfolded state energies . . . . . . . . . . . . . 33
3.2 Adaptive Monte Carlo simulations . . . . . . . . . . . . . . . . . . . 35
3.3 Biased holo simulation and analysis . . . . . . . . . . . . . . . . . . . 37
CONTENTS

9.3 Running the tutorial ........................................... 71

10 Adding D-amino acids at a specific position .............. 77

11 Using Toulbar2 for exact optimization ..................... 79

IV Solvent models in Proteus ................................. 81

12 Surface area calculations ................................. 83

12.1 Accessible Surface Area in protX ......................... 83

12.1.1 Syntax ................................................. 84

12.1.2 Example .............................................. 84

12.2 Approximate Fraternali or FFVG method .............. 84

12.2.1 Definitions ........................................... 84

12.2.2 Implementation in protX ............................. 85

12.3 Approximate LCPO method .............................. 87

12.3.1 Implementation in protX: the ESURF energy term ... 87

13 Nonpolar solvation ........................................... 89

13.1 Theory ..................................................... 89

13.1.1 Solute-solvent van der Waals dispersion model ... 89

13.1.2 Gaussian Nonpolar Solvent Model .................... 90

13.2 Implementation ........................................... 91

13.3 Syntax ..................................................... 93

13.3.1 Solute-solvent van der Waals dispersion energy ... 93

13.3.2 Setting up the parameters .......................... 94

13.3.3 Example: minimization and MD with GBDILK .... 95

14 Generalized Born electrostatics ........................... 97

14.1 Introduction .............................................. 97

14.2 Theory .................................................... 98
CONTENTS

17.1.1 Syntax ................................................. 127
17.1.2 Example: topology of a leucine ................. 128

17.2 Parameter Statement ................................ 129
17.2.1 Syntax ............................................. 129

17.3 Topology and parameter files .................... 133
17.3.1 Amber ff99SB and ff14SB ......................... 133
17.3.2 CHARMM “top_all22*” and “par_all22*” force field .... 133
17.3.3 AMBER/OPLS “tophopls.pro”, “parhopls.pro” files ...... 133
17.3.4 Files “toph19.sol” and “param19.sol” for TIP3P water .... 133

17.4 Generating the molecular structure .......... 133
17.4.1 Syntax ............................................. 134
17.4.2 Example: a polypeptide chain ................. 134

17.5 Patching the molecular structure .......... 135
17.5.1 Syntax ............................................. 135
17.5.2 Example: a disulfide bridge ................. 135

17.6 Deleting atoms ........................................ 136

17.7 Duplicating the Molecular Structure .......... 136

17.8 Structure statement ................................ 136

17.9 Writing a molecular structure file ........... 137

18 Energy function ........................................ 139

18.1 Empirical Energy Functions ..................... 139
18.2 Bonded terms ....................................... 139
18.3 Nonbonded energy terms ......................... 140
18.3.1 Van der Waals function ....................... 140
18.3.2 Electrostatic function ......................... 141
18.3.3 Intramolecular interactions ................. 141

18.4 Turning energy terms on or off ................. 142

18.5 Energy statement ................................... 142

18.6 Energy calculation between selected atoms ... 143
18.6.1 Syntax .................................................. 143

19 Geometric and energetic analysis ........................................ 145
  19.1 Analysis of conformational energy terms .......................... 145
  19.2 Analysis of the nonbonded energy terms ......................... 146

20 Cartesian coordinates ................................................. 149
  20.1 Coordinate statement ............................................ 149
  20.2 Rotamer implementation in protX ............................... 150
  20.3 Write coordinate statement ..................................... 151
  20.4 Building hydrogen positions ................................... 152

21 Coordinate restraints and constraints ................................ 153
  21.1 Harmonic coordinate restraints ................................ 153
  21.2 Dihedral restraints ............................................. 154
  21.3 Planarity restraints ............................................. 155
  21.4 Fixing atomic positions ........................................ 155
    21.4.1 Syntax .................................................. 156
  21.5 Fixing distances with SHAKE .................................. 156

22 Conjugate gradient energy minimization ............................ 157

23 Molecular dynamics .................................................. 159

List of protX statements ............................................... 161
Overview

Proteus has four components:

1. the molecular simulation program **protX**, mostly written in Fortran 90;

2. a set of scripts in the protX scripting language that control the calculation of an energy matrix for the system of interest [1];

3. a C program, **protMC** for exploring the space of sequences and conformations using various search algorithms, including Monte Carlo (MC);

4. a collection of perl, python, and shell scripts that automate various steps.

To obtain an overview and use this manual, the reader may want to first read the 2013 Proteus article (Simonson et al, J Comp Chem, 2013) [2], which includes details on the theoretical methods and the energy function.

We assume the reader has basic knowledge of Unix. The distribution files have been tested in a linux environment with an Intel processor and Intel compilers, although compilation should not be necessary for Intel-based machines, and using Gnu compilers should not be difficult.

This manual has five parts. **Part I** focusses on practical applications. We first describe the directory structure in the Proteus distribution and the main files used in applications. Then, we describe the steps in a typical application: system preparation, energy matrix calculation, searching sequence/conformation space, postprocessing and analysis. Finally, we describe and comment a series of tutorials provided with the distribution.

**Part II** focusses on the **protMC** program, which performs Monte Carlo exploration. It includes the complete set of options for Monte Carlo.

**Part III** describes installation and testing, along with several tasks of general interest, including automatically editing the energy matrix or modifying the rotamer library.

**Part IV** describes the implicit solvent models used by Proteus. This includes some theoretical background and implementation details, which involve both protX and protMC.

**Part V** provides documentation of the **protX** program for energy matrix calculation. Since protX uses the same command parser as Xplor, users can also use the Xplor documentation, which is more detailed. Part V includes a brief description of
the molecular mechanics model used for the energy matrix, excluding the solvation component (treated above).

Comments and bug reports

Please send comments, suggestions and bug reports to Thomas Simonson at Ecole Polytechnique:

thomas.simonson@polytechnique.fr
thomas.simonson@polytechnique.edu
Part I

Practical applications
Chapter 1

Overview of programs and procedures

1.1 Directory structure and files

1.1.1 Proteus source directories

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

- $CPD/doc: documentation files, including this manual: Proteus.pdf
- $CPD/tutorials: five Proteus tutorials
- $CPD/protMC: source code and executable for the Monte Carlo program protMC
- $CPD/rotamers: files that define the protein rotamer libraries
- $CPD/bin: auxiliary perl, python shell scripts
- $CPD/protX: top directory for the molecular modelling program protX
- $CPD/inp: protX scripts for system setup and energy matrix calculation
- $CPD/lib: protX macros or “stream files” for system setup and energy matrix calculation
- $CPD/protX/toppar: protX topology and parameter files
- $CPD/protX/src: source directory for protX; includes Makefile
- $CPD/protX/obj: protX object files and executable protX.exe

The most important files are described in the next sections.
1.1.2 User directories for a Proteus application

For a user running a given application, we define a top project directory, say $PROJ. This could be /home/dupont/PDZ. The subdirectory setup is partly imposed by the software, especially the matrix calculation. While complex, it is mostly created automatically. A typical setup would be:

- $PROJ/build: initial system setup for protX
- $PROJ/lib: local copy of the protX stream files that define the main parameters for the calculation; edit these files as needed
- $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: protX output files are written here
- $PROJ/matrix/err: protX 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 bsovldb
- $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/protMC: directory for the Monte Carlo simulations
- $PROJ/reconstruct: directory for 3D structure building and postprocessing
1.2 Using Proteus for a typical protein system

1.2.1 System preparation for the matrix calculation

Protein setup takes place in the build directory and starts from a PDB file, edited (eg, manually) so that the atom names conform to the conventions of the force field that will be employed, normally Amber ff99SB. The main model parameters are set by editing a single file, $PROJ/lib/parameters.str, which is written in the protX command language and where the user sets flags for the choice of force field, solvent model, and so on. The most important parameter in this file is the protein dielectric constant (used in combination with Generalized Born electrostatics). The other parameters can normally be left at their default values. The software location and project directories should be set by copying the file $CPD/bin/project.sh to the $PROJ/matrix directory and editing it. ProtX is run using a shell script $PROJ/build/build.sh which executes $PROJ/build/build.inp: protX < 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) \cite{3, 4}. If a single ligand is to be used, it can be created in build.inp and written to allh_protein.psf.

1.2.2 CPD setup: files to edit in $PROJ/lib

For the system build, above, only $PROJ/lib/parameters.str had to be edited. For the following steps, several files should be inspected and modified as needed:

- **parameters.str**: sets the force field, solvent model, dielectric constant, and other parameters (seen above)
- **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
- **phia.str**: sets the atomic surface energy coefficients; see recent papers \cite{5, 8}
- **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.
1.2.3 The energy matrix

A flowchart for the entire calculation is shown in Fig. 1.1.

**System preparation for CPD: the setup.inp step**  A second, more complex step starts from the generic build above and prepares the system specifically for a CPD calculation. It starts with a bash script, $PROJ/matrix/setup.sh. The user has already edited $PROJ/matrix/project.sh to define the software location. One should now edit the file $PROJ/lib/sele.str to choose which residues will be active (they mutate), inactive (flexible but don’t mutate), or frozen. The main task is performed by protX, which executes the script setup.inp. The active residues are modified by grafting all possible sidechain types onto their backbone $C_\alpha$. The resulting residues are referred to as “giant” residues.

![Flowchart](image)

**Figure 1.1: Flow chart for the energy matrix and sequence generation**

**System preparation for CPD: the setupI.inp step**  A second setup step is done by the bash script runI.sh. protX executes the script setupI.inp, which is ap-
1.2. USING PROTEUS FOR A TYPICAL PROTEIN SYSTEM

plied to each residue in the protein. 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. We also compute the solvation radii of the side chain atoms and store them in bsolv.pdb.

Diagonal matrix elements The runI.sh script goes on to compute the diagonal elements of the energy matrix. protX executes the script matrixI.inp, computing the interactions of each side chain $I$ with itself and the protein backbone. The $II$ diagonal matrix element is written to a file, matrix_I_I.dat.

Off-diagonal matrix elements For the off-diagonal matrix elements $IJ$, the calculations are controlled by a shell script runIJ.sh, which executes protX using the script matrixIJ.inp. This protX script loops over all pairs of active and inactive positions $I, J$ and all their types and rotamers. The interactions considered are those of side chain $I$ with side chain $J$. The final molecular mechanics energy and $IJ$ surface energy are written to the matrix file matrix_IJ_I_J.dat. At the end of runIJ.sh, the matrix elements are concatenated into a single file, which is then split into a diagonal (or backbone) file and an off-diagonal (or pairwise) file: matrix.dat, matrix.bb, matrix.pw, all in the dat subdirectory. The procedures for the $II$ and $IJ$ matrix elements are schematized below.

Procedure to compute II diagonal matrix element

```plaintext
define variable position I {
    get mutation space for position I
    if nativerot then handle position I native rotamers
    foreach amino acid type ti in mutation space I {
        get number of rotamers for amino acid ti
        foreach corresponding rotamer ri {
            position rotamer ri
            if gb then compute rotamer ri solvation radii
            minimize rotamer ri
            if gb then update rotamer ri solvation radii
            calculate ii energy matrix element
            write coordinates of rotamer ri
        }
    }
}
```
Procedure to compute IJ off-diagonal matrix element

foreach variable position I {
    get mutation space for position I
    foreach amino acid type ti in mutation space I {
        get rotamer space for type ti at position I
        foreach corresponding rotamer ri {
            read coordinates of rotamer ri
            foreach variable position J < I {
                if \( ij \) Cbeta distance below threshold then
                    get mutation space for position J
                    foreach amino acid type tj in mutation space J {
                        get rotamer space for type tj at position
                        foreach corresponding rotamer rj {
                            read coordinates of rotamer rj
                            if min dist sidei / sidej < 12 \( \text{A} \) then
                                if min dist sidei / sidej < 3 \( \text{A} \) then
                                    minimize rotamers ri, rj
                                end
                            end
                            calculate matrix element IJ
                        end
                    end
                end
            end
        end
    end
}
1.2.4 Exploring sequence/rotamer space with protMC

With the matrix in place, the sequence/rotamer exploration is done with protMC. A single command file controls the calculation, with an XML format and flexible commands. A simple example is given below; full examples are given in the Proteus tutorials. Details and a complete list of options are given in Chapter 5. Sequences are output as lists of rotamers, along with their folding energies. Rotamers are numbered using the internal protMC numbering, which identifies both amino acid type and rotamer. Conversion to a human-readable format is done by protMC in a postprocessing step. Several perl and python scripts are available ($CPD/bin) to compute sequence properties, such as similarity to a reference alignment. Reconstruction of 3D structures is described further on.

Figure 1.2: ProtMC command file

```xml
# proteus command file for an MC run
<Mode> MONTECARLO </Mode> # use MC for exploration
<Energy_Directory> ../matrix </Energy_Directory> # location of the matrix
<Temperature> 0.6 </Temperature> # kT in kcal/mol units
<Trajectory_Length> 1000000 </Trajectory_Length> # number of steps per run
<Seq_Output_File> prod.seq </Seq_Output_File> # output file for sequences
<Space_Constraints>
  4 ALA
  5 LEU
</Space_Constraints>
```
Chapter 2

Two test systems

2.1 The Syndecan-1 octapeptide

This is an 8-residue peptide (Fig. 2.1) taken from the C-terminus of the Syndecan-1 protein. We refer to the test directory (tuto_Sdc1/) as $PROJ. A README file (shown below) indicates the steps to follow. Model parameters are assigned in $PROJ/lib. The solvent model and other parameters are set in parameters.str. We use the Amber ff99SB force field with a simple but well-optimized GB variant [9, 10]. In sele.str, we set residues 4 and 5 to be “active”, meaning that they will mutate during the MC simulation. All other residues are “inactive”, meaning that they will explore rotamers but not mutate. The sequence/rotamer exploration is done by a Replica Exchange Monte Carlo run, with four replicas and ten million MC steps per replica (see MC.conf).

Figure 2.1: The Syndecan-1 octapeptide.
A) Build phase

0) Go into build directory
1) Prepare PDB file compatible with protX and Amber ff99SB atom names, call it model.pdb
2) Edit ../matrix/project.sh to adapt a few environment variables to your situation
3) Check files in lib subdirectory (or just accept the current default settings), especially sele.str and phia.str; also parameters.str (defaults should be OK)
4) Run this step by doing ./build.sh (from a bash shell)

B) Setup energy matrix calculation and compute matrix diagonal

1) Go into matrix subdirectory
2) Run setup and matrix diagonal: ./setup.sh then ./runI.sh [<nb_cpu>]
   Parallel execution if nb_cpu present and >1 (parallel bash command must be installed)

C) Off-diagonal energy matrix elements

1) Compute matrix by doing ./runIJ.sh [<nb_cpu>|<queue>] [<pair_list>]
   Optional arguments specify a particular pair list and a PBS queue

D) Perform Monte Carlo simulations

1) In protMC subdirectory, edit protMC .conf files (or accept default settings)
2) run MC and postprocessing with ./run.sh

E) Reconstruct structures

1) In reconstruct subdirectory, execute ./reconstruct.sh
   Reconstructed models are in reconstruct/pdb
2.1. THE SYNDECAN-1 OCTAPEPTIDE

The main directories and files are listed below, with a few comments:

<table>
<thead>
<tr>
<th>directory</th>
<th>Main files</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>build.inp, model.pdb, build.sh, build.out</td>
<td>the chain termini are unpatched, with dangling NH and CO</td>
</tr>
<tr>
<td>lib</td>
<td>sele.str, phia.str, parameters.str, mutation_space.dat</td>
<td>protX stream files, which set most of the model parameters; mutation space and reference energies are needed for active positions 4-5</td>
</tr>
<tr>
<td>matrix</td>
<td>setup.sh, runI.sh, runIJ.sh, dat/local</td>
<td>the matrix calculation is run from here; the matrix files are written in dat/</td>
</tr>
<tr>
<td>matrix/local</td>
<td>Bsolv/ Chis/ EnrFltr/ Mut/ Nbrot/ Rota/</td>
<td>these contain the position-specific information on allowed mutations, rotamers, and GB radii</td>
</tr>
<tr>
<td>protMC</td>
<td>run.sh, MC.conf, proteus.seq_N, proteus.rich_N</td>
<td>run.sh does everything; MC.conf is a protMC command file; proteus.seq_N and proteus.rich_N are designed sequences produced by replica N</td>
</tr>
<tr>
<td>reconstruct</td>
<td>run.sh, pdb/</td>
<td>Generate 3D structures from rotamer information in proteus.seq_0; run.sh does everything, using $CPD/inp/reconstruct.inp; PDB files are in pdb/</td>
</tr>
</tbody>
</table>

In the Proteus distribution, many but not all of the output files have been left in place. Output files for each replica are labelled by replica number. Thus, the coldest replica (replica 0) produces the files proteus.seq_0 (sequences expressed with protMC internal numbering), proteus.rich_0 (sequences expressed with amino acid types and residue numbers), and proteus.ener_0 (folding energies of each sequence). The file proteus.dat_0 is produced by the python script analyze_seq.py. It lists the sequences sampled by replica 0, by order of decreasing population (215 sequences), with energy statistics. Some designed structures are in $PROJ/reconstruct/pdb (1 structure each for the top 10 sequences). Some output is listed below.

Beginning of proteus.rich_0: first 3 states sampled by replica 0

```
> 1 backbone: (null)
SEQ/1 2 3 4 5 6 7 8
AA/ T K Q L D F Y A
ROT/ 8 9 15 6 3 1 8 1
> 5 backbone: (null)
AA/ T K Q F D F Y A
ROT/ 8 9 15 3 3 1 8 1
> 8 backbone: (null)
AA/ T K Q F D F Y A
ROT/ 9 9 15 3 3 1 8 1
```

Beginning of proteus.dat_0: most populated sequences (positions 4-5)
# AVE_ENERGY MIN_ENERGY MAX_ENERGY SEQUENCE COUNTS PROBA*100

-30.70  -43.27  -27.69   KK  6898145  68.98  
-31.44  -41.54  -28.82   KR  1737038  17.37  
-32.24  -42.04  -29.31   RK  592334   5.92   
-32.12  -42.29  -29.62   HK  268959   2.69   
-32.91  -41.85  -30.27   KH  168436   1.68   

Beginning of proteus.seq_0: 6 first states sampled by replica 0

#  residence  Replica temperature=0.6
#id  time  energy  ______Rotamers______
  1   1  -66.943  7 8 14 135 53 0 7 0
  5   1  -64.311  7 8 14 207 53 0 7 0
  8   2  -63.037  8 8 14 207 53 0 7 0
 14   1  -56.808  8 8 14 153 123 0 7 0
 16   3  -53.170  2 8 14 153 123 0 7 0

2.1.1 Common errors and suggestions for real projects

- Atom/residue name convention should be consistent with Amber ff99SB
- Missing topology/parameter(s) (for unusual molecules)
- Missing END at the end of the pdb file
- Path to a file longer than 80 characters (produces a protX error)
- In applications with a ligand, avoid starting the ligand name with a number
2.2 The Tiam1 PDZ domain

This is an example of whole protein design. $PROJ$ is the test directory tuto_PDZ/.
The 83-residue Tiam1 PDZ domain has Syndecan-1 as its biological ligand. All positions except Gly and Pro are allowed to mutate, into all types except Gly and Pro, as indicated in sele.str (where CYX designates cysteines engaged in a disulfide bond):

```plaintext
! Define active residue
vector ident (store2) (not (resn GLY or resn CYX or resn PRO))
```

Solvent is modeled with a sophisticated GB procedure, where the fluctuations of the dielectric boundary are treated explicitly. This procedure is referred to as “exact GB” or the “FDB” method, depending on the context. Exploration is done with Replica Exchange Monte Carlo, using four replicas. Replica 0 samples 92685 distinct sequences, listed in proteus.dat_0 by decreasing population:

-390.237 -394.099 -385.173
ERKTVQICLjQQTWKSLSYVSMQAYVjQJQNYYQCVIQTSWVSISRECSKLQKREAENNDKSETVELKVE 1517 0.02

-402.875 -406.711 -399.746
RWLTMLLjQSEQHQSQjEQVEQSAVVKVKEHjQKMVQIGLVCAYCQLVKEKVVFREDLSEHCEICR 1484 0.01

-393.556 -399.091 -390.653
EYQTVSIKCFMKRTVCYKFMKTTTVHjQQECYRAAIRCTCIASCTECAQLLYERNVSNHSTVELEKVR 1399 0.01

Etc

Notice that h, j, H designate the two singly- and the doubly-protonated states of His. Seven Gly and two Pro positions are not included in the output, so the output sequences are only 74 residues long. A sequence logo can be produced using the files in the seqlogo subdirectory. A simple version is shown, where Gly, Pro positions are excluded. To include Gly, Pro or to limit the logo to selected positions, an intermediate file (sorted.profile, one line per position) should be edited.

![Sequence logo](image)

Figure 2.2: A logo representing the designed Tiam1 sequences.
Chapter 3
Designing for binding with adaptive MC

This tutorial is more complex, and shows how to design positions in an enzyme to select for the binding affinity of a particular ligand. The tutorial was mostly written by Vaitea Opuu. Files are in the test directory adaptive_MC/. The methodology was presented in two recent articles [11, 12]. Proteus is currently the only CPD tool that allows to design directly for binding affinity and/or specificity on a large scale. The enzyme here is tyrosyl-tRNA synthetase (TyrRS) from Escherichia coli. The ligand is the unnatural amino acid azido-phenylalanine (azPhe). Three positions in the active site are allowed to mutate (they are active). In this tutorial, they are allowed to mutate into just a few types. The procedure has two main steps: an adaptive step, performed for the apo protein, where a bias potential is optimized such that all allowed sequences are sampled with comparable probabilities. Next comes a sampling step, where the protein:ligand complex is simulated using the bias from step 1. The bias effectively subtracts out the apo state, so that in the MC holo simulation, sequences are populated according to a Boltzmann distribution controlled by the binding free energy. As a result, tight binding sequences are exponentially enriched in the output.

This tutorial uses a GBLK implicit solvent model [8], where LK stands for Lazaridis-Karplus. This model is described in Part IV of this manual. The other tutorials all use a GBSA solvent. The GBLK model is still under examination for CPD, but it already appears to give results of comparable quality to GBSA for several benchmarks problems: protein stability mutations, scoring protein loop conformations, PDZ:peptide binding free energies [7], and aminoacyl-tRNA synthetase:substrate binding. Importantly, within the current Proteus release, it speeds up the energy matrix calculation by a factor of four. To use GBLK, we adjust some options in parameters.str (compare the file for this tutorial to one of the others).
During this tutorial, we will:

- Build the system in the *apo* (unbound) and *holo* (bound) states
- Pre-compute the energy matrix for each state
- Compute a set of unfolded energies or reference energies (with a simple tripeptide unfolded model)
- Adaptively learn an optimal bias for the *apo* state
- Compute the biased populations of sequences visited in the *apo* state
- Sample the *holo* state using this bias
- Analyze the results
- Reconstruct 3D structures for some variants

*The tutorial takes about 3 hours to complete.* We assume that the location of Proteus is defined by the environment variable `$CPD`, which could be something like `/usr/local/Proteus`.

### 3.1 Protocol to compute the matrix

#### 3.1.1 System parameters and build

The “build” step has to be done for **both the *apo* and the *holo* states**. We describe the *holo* case here. The *apo* case is nearly identical. In the *holo* work directory `$PROJ/holo`, go to the `lib` subdirectory. The environment variable `$MYLIB` is defined to be an alias of this directory, here and in the Proteus scripts. `$MYLIB` contains the files that configure the Proteus calculation, including a choice of mutating or active positions. These are set in the `sele.str` script. In this tutorial, active positions are 37, 126, 182, 183, 186. Positions nearby are flexible or inactive (they explore rotamers but do not mutate):

```
! Define active residue
vector ident (store2) ( segid A and ( resid 37 or resid 126 or resid 182 or resid 183 or resid 186 )

! Define inactive residues: positions with at least one side chain atom
! within 14 A of the ligand CZ (defined by its coordinates)
vector ident (store1) ((not (store2 or resn GLY or resn CYX or resn PRO or resn ACE)) and (byres (not (name CA or name N or name C or name O or name H*) and (point (10.8 172.4 244.6) around 14.)))
```
3.1. PROTOCOL TO COMPUTE THE MATRIX

Since the calculations include a ligand, the corresponding topology and parameter files must be available and included in the system setup, through the file $MYLIB/toppar.str, shown below:

```plaintext
topology
@@TOPPAR:amber/masses_parm99.rtf ! Masses
@@TOPPAR:amber/amino_parm99SB.bbunif.rtf ! protein topology
@@TOPPAR:amber/giant_parm99SB.rtf ! macros for mutations
@@MYLIB:azidophe.rtf ! topology file describing the ligand
end
parameters
@@TOPPAR:amber/parm99SB.GB.prm
@@MYLIB:azidophe.prm ! force field parameters for the ligand
end
```

The ligand is also present in the PDB file describing the initial protein:ligand complex, $PROJ/holo/build/model.pdb. Notice that for a new application and ligand, the user may need to develop her/his own force field parameters. In some applications (but not here), the energy function includes a Surface Area term: the ligand atoms are then assigned types in the phia_lig.str file (by analogy to those used for the protein, see phia.str). These types determine the surface coefficients used in the energy function [8, 10].

At this point, we can go to the $PROJ/holo/build directory and run

```
./build.sh
```

which executes protX using the script build.inp. Three files are output:

- build.out: protX log file
- allh_model.pdb: minimized pdb file
- allh_model.psf: so-called structure file (2D structure of the system)

### 3.1.2 Matrix calculation

The next task is the matrix calculation, which is actually a three-step process. At this step, it is necessary to have defined the allowed conformers or “rotamers” of the ligand. In this tutorial, the ligand is a tyrosine analog, with a simple side chain modification. Furthermore, we are interested in a single ligand pose, where the ligand backbone is in the same position as that of the natural Tyr ligand in the native complex (since we want the analog to act as a substrate of the enzyme).
Therefore, we have simply adapted the usual Tyr side chain rotamers to this case. The azPhe rotamers are defined in a collection PDB files (one per rotamer) located in `$PROJ/holo/ligrota/Rota`.

Go now to the subdirectory `$PROJ/holo/matrix`. The location of the ligand rotamers is defined by an environment variable in several bash scripts, as is the location of Proteus and of the present project. In any application, the user must make sure this information matches her/his actual situation. The bash scripts are indicated in the following steps.

**Step 2:** We now run a task that starts to prepare the matrix calculation:

```
./setup.sh
```

This runs protX with the script `setup.inp`. Output files are in `$PROJ/holo/matrix` and its subdirectories:

- `setup_nogiant.pdb`: 3D structure with extra information (atom burial, ...)
- `setup.pdb`: active positions now have multiple side chains
- `local/Bsolv/bsolv.pdb`: contains solvation radii for backbone atoms
- `position_list.dat`: list of positions in the system
- `dat/`: the matrix output directory has been created

To make the tutorial very fast, we recommend an optional step at this point (not needed for applications): the mutation spaces of the five active positions should be trimmed, so that only mutations to/from Ala are considered. This kind of restriction is normally applied further on, during the MC simulations. Doing it now will speed up the matrix calculation. Go to `matrix/local/Mut` and edit the files `37_active_TYR.dat`, `126_active ASN.dat`, ..., `186_active_LEU.dat`, so that only ALA and the native residue type appear.

**Step 3:** The next step is to run the command that computes the diagonal terms of the energy matrix:

```
./runI.sh
```

This runs protX with the scripts `setupI.inp` and `matrixI.inp`. Output files are in `tyrRS/holo/matrix` and its subdirectories, including:

- `dat/matrix_I_12.dat`, etc: diagonal matrix terms for residue 12, etc
- `pair_list.dat`: the list of residue pairs that will be computed (next)
3.1. PROTOCOL TO COMPUTE THE MATRIX

- `local/EnrFltr/548_<AA-name>.dat`, etc: list of allowed rotamers for residue 548, etc

- `local/Rota/548.pdb`, etc: Rotamer structures for residue 548, etc

**Step 4:** The final step is to run the calculation of off-diagonal matrix terms:

```
./runIJ.sh <NB cpu> pair_list.dat
```

For large systems, this step can take a lot of computational resources. The arguments are the number of processors or computer cores to use (on a multi-core machine) and the file containing the list of residue pairs. To use multiple cores, the gnu parallel package should be installed. For this tutorial, using 16 cores, the matrix calculation takes a few hours. The main output files are the elements of the matrix, such as

- `dat/matrix_IJ_10_12.dat`: off diagonal matrix elements for the residue pair 10, 12

Once the diagonal and off-diagonal calculations are complete, we can concatenate files in the `matrix/dat/` subdirectory in order to create two larger files, `matrix.bb` (diagonal terms) and `matrix.pw` (off-diagonal terms). **Step 5:**

```
$CPD/bin/concat_matrix.sh
```

The calculation of the energy matrix is now complete. The same steps 1–5 should now be done for the `apo` system, in `$PROJ/apo/`.

### 3.1.3 Computing the unfolded state energies

In this tutorial, the design will produce sequences based on the ligand affinity, without ever checking the stability or folding energy of the designed sequences. This can lead to unrealistic predictions, and a calculation of the folding energy is necessary as a sanity check (`apo` state only). We recall that the unfolded state does not require a 3D structural model, but relies on a set of unfolded or “reference” energies $E_{uf}(t)$. These determine the contribution of a single residue to the unfolded state energy. They depend on the side chain type but not the residue position within the polypeptide chain (as usual in CPD). Although complex procedures can be used to empirically parameterize the $E_{uf}(t)$, a simpler, less empirical method is used here that should be sufficient for affinity-based design. We simply compute the energy of each side chain type in the context of its own amino acid and the adjacent backbone groups. This is similar to popular “tripeptide” models of the unfolded state.

**Step 6:** In the `apo` subdirectory `$PROJ/apo/matrix/`, run the command:

```
erefI.sh
```
The reference energies are output in the files:

- **eref.conf**: reference energies for individual active positions
- **avg_eref.conf**: reference energies averaged over active positions

For our system, the resulting values are:

<table>
<thead>
<tr>
<th>amino acid</th>
<th>reference energy</th>
<th>amino acid</th>
<th>reference energy</th>
<th>amino acid</th>
<th>reference energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALA</td>
<td>7.54</td>
<td>GLU</td>
<td>-19.87</td>
<td>MET</td>
<td>0.90</td>
</tr>
<tr>
<td>ARG</td>
<td>-52.58</td>
<td>HID</td>
<td>12.84</td>
<td>PHE</td>
<td>16.62</td>
</tr>
<tr>
<td>ASH</td>
<td>-9.60</td>
<td>HIE</td>
<td>12.03</td>
<td>SER</td>
<td>-0.84</td>
</tr>
<tr>
<td>ASN</td>
<td>-17.07</td>
<td>HIP</td>
<td>20.05</td>
<td>THR</td>
<td>-2.94</td>
</tr>
<tr>
<td>ASP</td>
<td>-20.38</td>
<td>ILE</td>
<td>7.73</td>
<td>TRP</td>
<td>13.94</td>
</tr>
<tr>
<td>CYS</td>
<td>5.39</td>
<td>LEU</td>
<td>0.20</td>
<td>TYR</td>
<td>2.76</td>
</tr>
<tr>
<td>GLN</td>
<td>-16.42</td>
<td>LYS</td>
<td>-4.30</td>
<td>VAL</td>
<td>2.93</td>
</tr>
</tbody>
</table>
3.2 Adaptive Monte Carlo simulations

With the *apo* and *holo* matrices in place, we turn to the adaptive MC phase. We run MC for the *apo* system, to optimize a bias potential that will flatten the energy surface in sequence space, allowing all (or most) sequences to be sampled. The bias will then be used to sample the *holo* state.

Go to the $PROJ/apo/protMC subdirectory. MC will be run with the protMC program, controlled by a configuration file adapt.conf. This file indicates which mutations are allowed for positions that are active (37, 126, 182, 183, 186). In this tutorial, only mutations to/from Ala are allowed. Notice that residue 182 is Asp in the native protein; ASH represents the protonated form of Asp, chosen here:

```
# Mutations to alanine only
<Space_Constraints>
37 ALA TYR
126 ALA ASN
182 ALA ASH
183 ALA PHE
186 ALA LEU
</Space_Constraints>
```

The form of the bias potential is specified by the following commands:

```
<Adapt_Space>
37-37
126-126
182-182
183-183
186-186
</Adapt_Space>
```

These commands indicate that bias terms will involve all five positions, but will only include “diagonal” terms; we do not use pairwise bias terms that involve two positions. For examples of pairwise biases, see Part II, below. Some other options are included in adapt.conf, to control details of the adaptation protocol:

```
<Adapt_Mono_Period> 1000 </Adapt_Mono_Period>
<Adapt_Output_Period> 10000 </Adapt_Output_Period>
<Adapt_Output_File> bias.dat </Adapt_Output_File>
```

During the adaptation simulation, it is best to include reasonable values for the unfolded energies; thus adapt.conf includes the values from Table 3.1.3.
At this point, **Step 7**, we run protMC in the $PROJ/apo/protMC$ subdirectory:

```
$CPD/protMC/protMC.exe < adapt.conf > adapt.log
```

Output files are:

- **bias.dat**: evolution of the bias during the trajectory
- **proteus_adapt.seq**: visited sequences
- **output.ener**: the energy of visited sequences

At the end of the adaptive procedure, we copy the final value of the bias from **bias.dat** into a new file, **bias.in**. The bias values are:

<table>
<thead>
<tr>
<th>position</th>
<th>type</th>
<th>bias</th>
<th>position</th>
<th>type</th>
<th>bias</th>
<th>position</th>
<th>type</th>
<th>bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>ALA</td>
<td>0.000</td>
<td>182</td>
<td>ALA</td>
<td>0.000</td>
<td>186</td>
<td>ALA</td>
<td>0.000</td>
</tr>
<tr>
<td>37</td>
<td>TYR</td>
<td>29.797</td>
<td>182</td>
<td>ASH</td>
<td>4.268</td>
<td>186</td>
<td>LEU</td>
<td>1.572</td>
</tr>
<tr>
<td>126</td>
<td>ALA</td>
<td>0.000</td>
<td>183</td>
<td>ALA</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>ASN</td>
<td>10.675</td>
<td>183</td>
<td>PHE</td>
<td>38.817</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An extended simulation of the *apo* system using the bias can be done, **Step 8**, with the protMC script **MC.conf**:

```
$CPD/protMC/protMC.exe < MC.conf > MC.log
```

**MC.conf** includes a statement:

```
<Bias_Input_File> bias.in </Bias_Input_File>
```

Postprocessing with **POST.conf**, **Step 9**, produces the sequences in human readable, “rich” format, **proteus.rich**:

```
$CPD/protMC/protMC.exe < POST.conf > POST.log
```

Finally, the populations of the visited sequences can be obtained, **Step 10**:

```
analyze_seq.py proteus.seq proteus.rich <nb_steps> ../matrix/active_list | proteus.dat
```

They are shown below in the form of a logo, obtained with (left) or without (right) the bias:
3.3 Biased holo simulation and analysis

3.3.1 Biased holo simulation

We go now to the $\text{PROJ/holo/protMC}$ directory. Simulating the holo system with the apo bias will now lead to sequences that are populated according to their azPhe binding free energy (sic). The command files MC.conf and POST.conf already used for the apo system can be used without modification:

Step 11: $\text{CPD/protMC/protMC.exe < MC.conf > MC.log}$
Step 12: $\text{CPD/protMC/protMC.exe < POST.conf > POST.log}$
Step 13: analyze_seq.py proteus.seq proteus.rich <nb_steps> \
        ../matrix/active_list > proteus.dat

Affinity-based sampling is finished. Sequence populations will now lead directly to binding affinities: see next section. The sampled sequences are shown below as a logo.

3.3.2 Affinity and stability estimations

Above, we computed sequence populations in the apo and holo states in the presence of the bias (apo and holo proteus.dat files). For two sequences $s$ and $r$ sampled in both states, we denote $p'_s$, $p'_r$ the biased holo populations and $p_s$, $p_r$ the biased apo populations. We can obtain the binding free energy difference as

$$
\Delta G_s - \Delta G_r = -kT \ln \frac{p'_s}{p'_r} - kT \ln \frac{p_s}{p_r}
$$

(3.1)
This is implemented in a python script, [Step 14:]

```bash
../affinity.py ../apo/proteus/proteus.dat proteus.dat bias.in \ 
   -rf YNdFL -p 37 126 182 183 186
```

The strongest affinities are listed in Table 3.2 relative to the wildtype sequence, taken as a reference. Notice that ‘d’ stands for protonated Asp. The estimated folding energy of each variant is also indicated. The relative stabilities were obtained by comparing populations in the biased apo state, and removing the difference in bias energies. The most stable variant is AAAAA.

<table>
<thead>
<tr>
<th>sequence</th>
<th>affinity</th>
<th>stability</th>
<th>sequence</th>
<th>affinity</th>
<th>stability</th>
<th>sequence</th>
<th>affinity</th>
<th>stability</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANAFL</td>
<td>-4.42</td>
<td>-42.30</td>
<td>ANdAA</td>
<td>-1.67</td>
<td>-75.63</td>
<td>AAdFL</td>
<td>-0.44</td>
<td>-48.65</td>
</tr>
<tr>
<td>ANAFA</td>
<td>-4.07</td>
<td>-43.76</td>
<td>ANAAL</td>
<td>-1.64</td>
<td>-80.69</td>
<td>AAAAL</td>
<td>-0.25</td>
<td>-90.67</td>
</tr>
<tr>
<td>AAAFL</td>
<td>-2.94</td>
<td>-52.19</td>
<td>ANdFA</td>
<td>-1.53</td>
<td>-37.23</td>
<td>AAdFA</td>
<td>-0.08</td>
<td>-50.06</td>
</tr>
<tr>
<td>AAAFA</td>
<td>-2.59</td>
<td>-53.55</td>
<td>ANAAA</td>
<td>-1.50</td>
<td>-82.59</td>
<td>AAAA</td>
<td>0.00</td>
<td>-92.46</td>
</tr>
<tr>
<td>ANdFL</td>
<td>-2.54</td>
<td>-34.81</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.3.3 Reconstruction

3D structure models are computed from the rotamer information by the bash script reconstruct.sh in the $PROJ/holo/reconstruct directory. The structures for YNdAL and ANdAL are shown below.

### 3.3.4 Suggestions

The mutation space here was very limited. When using larger, more realistic mutation spaces, one should increase the trajectory length (at least 1000 times the size of the combinatorial space). Another way to improve the MC simulation is to use replica exchange MC; see Part II [13]. In this tutorial, we used the simplified GB NEA model for the solvent. One can use a more accurate treatment, the GB FDB method [5], which increases the CPU time for the MC simulations by as much as a factor of 4. If FDB is used only in the binding site, the increase will be smaller, less than a factor of 2. Notice that FDB increases the cost of the matrix calculation only negligibly.
3.3. BIASED HOLO SIMULATION AND ANALYSIS

Figure 3.2: 3D structure of designed TyrRS mutant ANAFL with bound AzPhe (AZF). Mutated positions are red. Cross-eyed stereo view.

3.3.5 Testing selected variants with molecular dynamics simulations

A good way to test designed variants is to run molecular dynamics simulations (MD) with an explicit solvent model. This step can easily be applied to a few dozen variants before going on to experimental testing, which is essential but more expensive. Therefore, we recommend using MD as an additional computational filter to help reduce the number of variants proposed for experimental tests. Here, we show briefly how to take a variant produced with Proteus and prepare it for explicit solvent MD with the NAMD simulation program [14]. NAMD runs efficiently on inexpensive GPU computers and is available in many supercomputer centers. A 60 or 80 ns simulation can be run in a day on a GPU processor with NAMD, for example. The necessary files are included in the tutorial (subdirectory MD).

1. The first step is to run 3D structure reconstruction for a variant of interest, as explained above.

2. Create a directory for the MD; here we use holo/MD. Copy the PDB file and the psf file produced by the reconstruction to this directory. These would be in the directory reconstruct/pdb/ and named something like: rec.0.0.pdb and rec.0.0.psf. Rename them protein.pdb and protein.psf.
3. Execute a bash script that truncates the protein to a roughly spherical shape and solvates it:

`solvate.sh`

During MD, the outer portion of the truncated protein will be held in place by weak harmonic restraints, while the inner part (near the ligand) moves freely. The bash script also edits the final psf file, to make it compatible with NAMD. Indeed, with protX, some dihedrals appear multiple times in the psf, whereas NAMD expects unique dihedrals. This editing is done with a bash script and an awk script (`$CPD/bin/dihe_mult.awk`), both executed by solvate.sh.

4. The system is now ready for equilibration then production with NAMD. An example bash script is provided: `dyna.sh`.

5. Reread the produced trajectory (using NAMD or protX or charmm or Xplor) and extract interesting features, like 3D structures or rms deviations relative to the starting or wildtype complex; see the NAMD or Xplor manual or on-line tutorials.
Chapter 4

Acid/base calculations

This tutorial shows how to compute acid/base constants, or pK\(_a\)’s with Proteus. It was written by Francesco Villa and Savvas Polydorides. Files are in the test directory tuto_pKa/. The methodology was presented in two recent articles [5, 15]. Notice that another, different method was also published recently [16], but is not included in this tutorial. The method used here involves running MC simulations at a series of pH values. Selected residues (Asp, Cys, Glu, His, Lys, Tyr) are allowed to “mutate” by changing their protonation state. The associated energy change depends on the pH, as explained below. As pH increases, the deprotonated forms become more populated. By fitting population curves, the pK\(_a\) of each titratable side chain is estimated. The test protein is BPTI, which has 58 amino acids and 12 titratable groups. A README file recalls the main steps to follow, which are described briefly below.

4.1 System build and setup

The build, Step 1, is done as usual, in the tuto_pKa/build directory. For the setup step, positions that are allowed to titrate are set to be active (lib/sele.str). The more rigorous, FDB GB variant is chosen (lib/sele.str):

! exact-GB pairs will be defined as store4--store4 pairs
vector ident (store4) (all)

The protein dielectric constant is set to 4 (lib/parameters.str) The mutation space is initially the default space. Step 2: We execute setup.sh in tuto_pKa/matrix; the protX log file is out/setup.out. We then go into matrix/local/Mut and edit the files corresponding to the titrating positions. Asp positions are allowed to have the types ASP and ASH; His positions are allowed to have the types HID, HIE, HIP, and so on. The matrix can now be calculated, in the matrix directory, by
executing \textbf{Step 3:} runI.sh, then \textbf{Step 4:} runIJ.sh. Multiple cores should be used if possible. protX log files are in the subdirectory out. The diagonal and off-diagonal matrix blocks are in matrix/matrix.bb and matrix/matrix.pw.

### 4.2 Running the pH scan

The pH scan is now performed in tuto\_pKa/titration, \textbf{Step 5} by executing titration.sh. A MC simulation is run at successive pH steps. For each pH value, the reference energy of the titrating residue types is adjusted by adding or subtracting a contribution $kT \ln 10 \approx 1.35$ pH. The sequences output by the MC are converted, \textbf{Step 6} to the rich format and analyzed, \textbf{Step 7} with analyze\_seq.py to produce populations. Finally, \textbf{Step 8} the populations corresponding to the individual titratable groups are collected in files like position10.dat and a perl script evalpKa.pl fits these to a standard titration curve and reports the pK$_a$ value and Hill’s coefficient in results.dat. The locations of selected directories and files for the titration steps are indicated below, relative to tuto\_pKa/titration:

<table>
<thead>
<tr>
<th>Selected directories and files for titration, relative to tuto_pKa/titration</th>
</tr>
</thead>
<tbody>
<tr>
<td>titration.sh</td>
</tr>
<tr>
<td>toolbox</td>
</tr>
<tr>
<td>conf</td>
</tr>
<tr>
<td>conf/mc.template.conf</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>prod</td>
</tr>
<tr>
<td>dat</td>
</tr>
<tr>
<td>dat/distr_7.0.dat</td>
</tr>
<tr>
<td>dat/position10.dat</td>
</tr>
<tr>
<td>results.dat</td>
</tr>
</tbody>
</table>

Some of the data are shown below:

**Top of dat/distr\_7.0.dat: sequences sampled at pH = 7.0**

<table>
<thead>
<tr>
<th>Emean</th>
<th>Emax</th>
<th>Emin</th>
<th>sequence</th>
<th>counts</th>
<th>%proba</th>
</tr>
</thead>
<tbody>
<tr>
<td>-35.93</td>
<td>-32.02</td>
<td>-39.838</td>
<td>RDFLEYTKARIIRyFYNALKALQTFVYRAKRNFKSAEDMRTA</td>
<td>1024</td>
<td>0.010</td>
</tr>
<tr>
<td>-39.78</td>
<td>-33.27</td>
<td>-48.925</td>
<td>RdFLEYTKARIIRyFYNALKALQTFVYRAKRNFKSAEDMRTA</td>
<td>3722</td>
<td>0.037</td>
</tr>
<tr>
<td>-38.40</td>
<td>-35.05</td>
<td>-43.889</td>
<td>RDFLEYTKARIIRyFYNALKALQTFVYRAKRNFKSAEDMRTA</td>
<td>2176</td>
<td>0.021</td>
</tr>
</tbody>
</table>
For each sequence, the number of visits (counts) is reported; each visit typically samples a different set of side chain rotamers. The mean, maximum, and minimum energies (Emean, Emax, Emin) are taken over all the visits and rotamer states. Sequence probabilities are in %. 18 sequences were sampled in this particular simulation. In the most populated state (97.57%), all the side chains are seen to have their standard physiological protonation state.

Part of dat/position10.dat: Tyr10 protonated fraction (%) vs. pH

<table>
<thead>
<tr>
<th>pH</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.00</td>
<td>0.997</td>
</tr>
<tr>
<td>7.50</td>
<td>0.990</td>
</tr>
<tr>
<td>8.00</td>
<td>0.974</td>
</tr>
<tr>
<td>8.50</td>
<td>0.931</td>
</tr>
<tr>
<td>9.00</td>
<td>0.832</td>
</tr>
<tr>
<td>9.50</td>
<td>0.640</td>
</tr>
<tr>
<td>10.00</td>
<td>0.423</td>
</tr>
<tr>
<td>10.50</td>
<td>0.239</td>
</tr>
<tr>
<td>11.00</td>
<td>0.133</td>
</tr>
<tr>
<td>11.50</td>
<td>0.073</td>
</tr>
</tbody>
</table>

The pKₐ is close to 9.75. In titration/results.dat, we see the fitted value is 9.88:

<table>
<thead>
<tr>
<th>resid</th>
<th>pKa</th>
<th>Hill’s coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3.00</td>
<td>0.75</td>
</tr>
<tr>
<td>7</td>
<td>4.00</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>9.88</td>
<td>0.75</td>
</tr>
<tr>
<td>15</td>
<td>10.50</td>
<td>1.00</td>
</tr>
<tr>
<td>21</td>
<td>10.69</td>
<td>0.81</td>
</tr>
<tr>
<td>23</td>
<td>11.50</td>
<td>1.00</td>
</tr>
<tr>
<td>26</td>
<td>10.50</td>
<td>1.00</td>
</tr>
<tr>
<td>35</td>
<td>9.00</td>
<td>0.75</td>
</tr>
<tr>
<td>41</td>
<td>11.75</td>
<td>0.88</td>
</tr>
<tr>
<td>46</td>
<td>11.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
The computed titration curves are shown in Fig. 4.1.

Figure 4.1: BPTI titration curves from the present method (grey) and an alternative method (black); reproduced from Villa & Simonson (2018) *Journal of Chemical Theory and Computation*, 14:6714.
Part II

The protMC program
Chapter 5

The protMC program

5.1 Overview

ProtMC is a C program that reads the energy matrix computed with protX, then explores a space of sequences and conformations. We mainly use Monte Carlo or Replica Exchange Monte Carlo (REMC), which generate Boltzmann ensembles. However, a heuristic, multi-start minimization can also be used, and is quite effective at locating the Global Minimum Energy Conformation, or GMEC for small and medium-size problems [13]. Importantly, protX can perform adaptive Wang-Landau MC, where the energy landscape is flattened thanks to a bias potential, to enhance sampling.

ProtMC is controlled by a command file, with an xml format. For REMC, on a multi-core machine, protX uses a shared-memory, OpenMP parallelization to increase speed. Sequences are output in the form of lists of rotamers, along with their folding energies. Rotamers are numbered using the internal protMC numbering, which identifies both amino acid type and rotamer. Conversion to a more verbose, human-readable format (Fig. 5.1) is done by protMC in a postprocess step. A series of perl and python scripts are available to compute sequence properties, such as similarity to a reference alignment.

The basic Monte Carlo move in Proteus (and CPD in general) is shown in Fig. 5.2: a mutation is performed in the folded protein, while the inverse mutation is done in the unfolded protein. Equivalently, the move can be seen as unfolding the starting variant (pre-mutation), while refolding the new variant (with the mutation). Thus, a mutation move, while ostensibly involving sequence space, actually takes place in a conformation space, where standard statistical mechanics apply. The simulation is said to produce a Markov chain of states. It leads to the same distribution of states as a macroscopic, equilibrium, physical system where all sequences S, S', ... are present at equal concentrations, and are distributed between their folded and unfolded states.
according to their relative stabilities. This is exactly the experimental system we want our simulation to mimic \[13\].

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
\text{id} & \text{time} & m(G1+G2) & G1[5] & G2[1] & \text{Temperature}=0.65 \\
\hline
0 & 2 & -70.12 & 180 & 132 & 3 & 147 & 2 & 132 \\
1 & 3 & -70.64 & 179 & 133 & 3 & 147 & 2 & 132 \\
\hline
\end{array}
\]

Figure 5.1: ProtMC sequence output files: raw format (upper two panels) and rich format (bottom panel).

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{id} & m(G1+G2) & G1 & G2 & \text{Temperature}=0.65 \\
\hline
0 & -70.12 & -56.19 & -13.94 \\
1 & -70.64 & -56.70 & -13.94 \\
\hline
\end{array}
\]

\[> \text{0} \quad \text{backbone: (null)}\]

\[\text{AA/ F K L K D K} \]

\[\text{SEQ/ 489 490 491 492 493 490} \]

\[\text{ROT/ 3 21 4 36 3 21} \]

Figure 5.2: A MC mutation move: a point mutation is performed in the folded state, along with the inverse mutation in the unfolded state.

5.2 Dictionary of protMC commands

The full set of protMC commands is listed in Table 5.1.
Table 5.1: protMC commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapt_Space</td>
<td>Define residue space to adaptively flattening</td>
</tr>
<tr>
<td>Adapt_Mono_Period</td>
<td>When to update single-position biases</td>
</tr>
<tr>
<td>Adapt_Pair_Period</td>
<td>When to update two-position biases</td>
</tr>
<tr>
<td>Adapt_Mono_Speed</td>
<td>Helps control bias increments</td>
</tr>
<tr>
<td>Adapt_Pair_Speed</td>
<td>Helps control bias increments</td>
</tr>
<tr>
<td>Adapt_Mono_Height</td>
<td>Helps control bias increments</td>
</tr>
<tr>
<td>Adapt_Pair_Height</td>
<td>Helps control bias increments</td>
</tr>
<tr>
<td>Adapt_Output_File</td>
<td>Where to write bias values</td>
</tr>
<tr>
<td>Adapt_Pair_Offset</td>
<td>When to start bias updates</td>
</tr>
<tr>
<td>Adapt_Mono_Offset</td>
<td>When to start bias updates</td>
</tr>
<tr>
<td>Adapt_Output_Period</td>
<td>How often to output bias</td>
</tr>
<tr>
<td>Backbone_Proba</td>
<td>Define characteristics of a backbone MC move</td>
</tr>
<tr>
<td></td>
<td>(when using multi-backbone MC)</td>
</tr>
<tr>
<td>Bias_Input_File</td>
<td>Input file containing a bias potential</td>
</tr>
<tr>
<td>Cycle_Number</td>
<td>Number of heuristic cycles for HEUR mode</td>
</tr>
<tr>
<td>Dielectric_Parameter</td>
<td>A scaling factor that divides electrostatic energy.</td>
</tr>
<tr>
<td>Energy_Output_File</td>
<td>Where to write energies</td>
</tr>
<tr>
<td>Energy_Directory</td>
<td>Where to find energy matrix (.bb, .pw)</td>
</tr>
<tr>
<td>Fasta_File</td>
<td>Output file for sequences in rich format</td>
</tr>
<tr>
<td>GB_BMAX</td>
<td>A threshold for GB solvation radii</td>
</tr>
<tr>
<td>GB_Method</td>
<td>NEA or FDB</td>
</tr>
<tr>
<td>GB_Neighbor_Threshold</td>
<td>Threshold for GB neighbor relation</td>
</tr>
<tr>
<td>Group_Definition</td>
<td>Define groups</td>
</tr>
<tr>
<td>Initial_Weights</td>
<td>Initialize state probabilities in mean field mode</td>
</tr>
<tr>
<td>Label</td>
<td>Define a name or alias for a set of positions</td>
</tr>
<tr>
<td>Lambda_Parameter</td>
<td>Mean Field relaxation parameter</td>
</tr>
<tr>
<td>Mode</td>
<td>Determines what task is done:</td>
</tr>
<tr>
<td></td>
<td>HEUR, MC, ADAPT, POSTPROCESS or mean field</td>
</tr>
<tr>
<td>Neighbor_Threshold</td>
<td>Energy threshold for MC neighbor definition</td>
</tr>
<tr>
<td>Optimization_Configuration</td>
<td>Define the energy function using groups and weights</td>
</tr>
<tr>
<td>Position_Weights</td>
<td>Probability to pick a position for a given MC move type</td>
</tr>
<tr>
<td>Print_Threshold</td>
<td>Energy threshold to limit the size of output files</td>
</tr>
<tr>
<td>Print_BSolv</td>
<td>Output GB solvation radii</td>
</tr>
<tr>
<td>Protein_Dielectric</td>
<td>Specify protein dielectric constant (FDB method needs it)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random_Generator</td>
<td>Choose random number routine within GSL library</td>
</tr>
<tr>
<td>Ref_Ener</td>
<td>Define reference or unfolded energies</td>
</tr>
<tr>
<td>Replica_Number</td>
<td>Number of replicas for REMC</td>
</tr>
<tr>
<td>Reset_Energies</td>
<td>Frequency to recompute energy from scratch during MC</td>
</tr>
<tr>
<td>Rseed_Definition</td>
<td>Choose random number seed</td>
</tr>
<tr>
<td>Space_Constraints</td>
<td>Restrict sequences or rotamers, or link two positions</td>
</tr>
<tr>
<td>Seq_Input</td>
<td>Specify a starting sequence/conformation</td>
</tr>
<tr>
<td>Seq_Input_File</td>
<td>Specify file to read a starting sequence/conformation</td>
</tr>
<tr>
<td>Seq_Output_File</td>
<td>Sequence output file (raw .seq format)</td>
</tr>
<tr>
<td>Sequence_Pass_Number</td>
<td>Maximum passes over the sequence per heuristic cycle</td>
</tr>
<tr>
<td>Solv_Neighbor_Threshold</td>
<td>Another energy threshold to define GB neighbors</td>
</tr>
<tr>
<td>Step_Definition_Proba</td>
<td>Define move probabilities for each MC step</td>
</tr>
<tr>
<td>Surf_Ener_Factor</td>
<td>Factor that multiplies surface energy term</td>
</tr>
<tr>
<td>Swap_Period</td>
<td>Period for exchanging temperatures between replicas</td>
</tr>
<tr>
<td>Temperature</td>
<td>For MC or Mean Field; multiple values if REMC</td>
</tr>
<tr>
<td>Trajectory_Length</td>
<td>Length of an MC trajectory (number of steps)</td>
</tr>
<tr>
<td>Trajectory_Number</td>
<td>Number of MC trajectories to run</td>
</tr>
<tr>
<td>Weight_Exchange_File</td>
<td>Probabilities for backbone exchange MC moves</td>
</tr>
<tr>
<td></td>
<td>(in multi-backbone MC)</td>
</tr>
</tbody>
</table>
We now describe each command briefly, in alphabetic order. Default values are given where appropriate (in brackets).

- **Adapt_Space**: This is the essential step to define which positions are flattened in Adapt mode. Individual positions are listed, say I, J, K, and possibly pairs of the form IJ, IK, JK. See adaptive MC tutorial in Part I for details.

- **Adapt_Mono_Period [5000]**: The frequency for updating the single-position bias terms (II, JJ, KK terms).

- **Adapt_Pair_Period [5000]**: The frequency for updating the two-position bias terms (IJ, IK, JK terms, when present).

- **Adapt_Mono_Speed [50]**: Bias increments have the form $\delta B_I = h e^{-B_I/E_0}$ [11]; here we set $E_0$ for the single-position terms.

- **Adapt_Pair_Speed [50]**: Set $E_0$ for the two-position bias terms.

- **Adapt_Mono_Height [0.2]**: Set the bias parameter $h$ for single-position terms.

- **Adapt_Pair_Height [0.2]**: Set the bias parameter $h$ for two-position terms.

- **Adapt_Output_File [adapt_out.dat]**: File where the bias values are written, at the end of every period.

- **Adapt_Mono_Offset [0]**: The step number where the first period begins for the single-position biases.

- **Adapt_Pair_Offset [infinity]**: The step number where the first period begins for the two-position biases.

- **Adapt_Output_Period [infinity]**: Set the period for writing the bias values to a chosen file.

- **Backbone_Proba**: Define characteristics of a backbone MC move, during multi-backbone MC [17]: the relative probability of a backbone move, the number of relaxation paths, their length.

- **Bias_Input_File [none]**: Causes an existing bias potential to be read from a file (usually the final value reached at the end of a previous adaptation run). The bias format is analogous to that of the matrix files.

- **Cycle_Number [100000]**: The number of heuristic cycles performed in HEURISTIC mode (multi-start minimization exploration method) [13].
CHAPTER 5. THE PROTM C PROGRAM

- Dielectric_Parameter [1.0]: This value divides the electrostatic energy term. It is mainly useful in the context of a simple CASA solvent model [10, 13].

- Energy_Output_File [output.ener]: File to output energy data from an MC trajectory or a heuristic search. With REMC, multiple files are output, one per replica; replica N has a trailing _N in the output file name.

- Energy_Directory [:]: The directory containing the energy matrix files to read, matrix.bb and matrix.pw. Default is current directory (where protMC is executed).

- Fasta_File [output.rich]: Output file for sequences in rich, Fasta-like format. Produced by the POSTPROCESS mode.

- GB_BMAX [10.0]: A threshold for GB solvation radii (to increase efficiency within protMC).

- GB_Method [False]: Activate FDB GB method if true; if false or absent, NEA is assumed.

- GB_Neighbor_Threshold [0.0]: A threshold that defines which “distant” positions are omitted from calculation of each residue’s solvation radius (for efficiency).

- Group_Definition: Define a group of residues.

- Initial_Weights [1, 1, ...]: Initialize state probabilities in MEANFIELD mode.

- Label: Give a name to a group of residues.

- Lambda_Parameter [1.0]: Mean field relaxation parameter [10].

- Mode [INFO]: The essential command that defines what calculation to do: HEUR, MC, ADAPT, POSTPROCESS, INFO or MEANFIELD. The INFO mode simply prints out information on the system.

- Neighbor_Threshold [0]: Interaction energy threshold that defines which positions participate in two-position MC moves. Default value 0 means that all pairs can make two-position moves.

- Optimization_Configuration: Modify the energy function using groups and weights.

- Position_Weights [uniform]: Increase or decrease the MC move probabilities for selected positions.
• **Print\_Threshold** [infinity]: Define an energy threshold, and only print out states within this threshold of the current lowest energy sampled so far. This is used to limit the size of output files.

• **Print\_BSolv** [0]: How frequently to write out the GB solvation radii of all residues. Default is 0, meaning the radii are never printed out.

• **Protein\_Dielectric** [4.0]: Indicate the value of the protein dielectric constant (needed for the FDB method; should match the value used in the matrix calculation).

• **Random\_Generator** [mt19937]: Specify a random number generator; must be available within the Gnu Scientific Library (GSL). If not specified, or if GSL is not installed, this will default to a reasonable routine, mt19937 from GSL.

• **Ref\_Ener** [0.0]: Specify unfolded energies for all or selected positions.

• **Reset\_Energies** [100]: The frequency to recompute energies from scratch.

• **Rseed\_Definition**: Choose an integer value for the random number seed (allowing one to (re)start a trajectory in a controlled and reproducible way). If unspecified, the current time is used.

• **Space\_Constraints**: Impose certain types or rotamers at certain positions. Can also be used to link positions, so that they always share the same residue type (i.e., they mutate together).

• **Seq\_Input**: Specify an existing state to use, e.g., to restart a trajectory.

• **Seq\_Input\_File**: Specify a file from which an existing state is to be read and used.

• **Seq\_Output\_File** [output.seq]: Indicate the file where the trajectory of sequences will be written. With REMC, multiple versions of the file will be written, one per replica, with a trailing \_N added to the file name, where N is a replica number.

• **Sequence\_Pass\_Number** [500]: In HEUR mode, heuristic cycles are run, where each cycle passes through the sequence multiple times and tries to improve the energy. This command specifies the maximum number of passes to do per heuristic cycle.

• **Solv\_Neighbor\_Threshold** [0.0]: An energy threshold to define positions that are treated as neighbors in the FDB GB method (used to exclude distant interactions, for efficiency).
• Step_Definition_Proba [Rot 1.0]: Provide the detailed set of move probabilities for MC.

• Surf_Ener_Factor [1.0]: A factor that multiples the surface energy term (used eg for tuning or parameterizing).

• Swap_Period [infinity]: The frequency with which to attempt swaps between replicas in REMC. The default is no swaps.

• Temperature [0.65]: Expressed as the thermal energy $kT$ in kcal/mol units. Used for MC or REMC. With REMC, one value per replica is needed.

• Trajectory_Length [$10^6$ steps]: MC trajectory length in steps (with REMC, this is the length for each replica).

• Trajectory_Number [1]: The number of MC trajectories to run (usually one).

• Replica_Number [1]: Number of replicas for REMC.

• Weight_Exchange_File: A set of probabilities for backbone exchange moves in multi-backbone MC [17].

## 5.3 Selected options for Monte Carlo exploration

**Monte Carlo mode**  The `<Mode>` command should be the first command in the .conf command file. Choosing Monte Carlo leads to MC sampling, with one or several replicas.

```
<Mode>
MONTECARLO
</Mode>
```

For replica exchange, one should specify the number and temperature of replicas and the frequency for attempting temperature swaps:

```
<Walker_Number>
4
</Walker_Number>
<Temperature>
0.6
0.9
1.3
1.8
```
Monte Carlo move probabilities Several options control the other MC move probabilities, which can be rotamer or type changes at one or two positions; for example:

```
<Step_Definition_Proba>
Rot 1.0
Rot Rot 0.1
Mut 0.2
Mut Mut 0.1
</Step_Definition_Proba>
```

The probabilities above do not add up to one; they will be normalized to have a total sum of one. For two-position moves, the 2nd position is chosen close to the first one, based on an interaction energy threshold (set by the Neighbor_Threshold option). Selected positions can be assigned increased move probabilities by applying weights:

```
<Position_Weights>
Rot 489 0.5
Rot 495 490-493 0.05
Mut 489-491 0.05
</Position_Weights>
```

These weights imply that for rotamer moves, position 489 will be chosen half of the time and all other positions half of the time. For mutation moves, position 489 will be chosen 1/20 of the time.

Exploration constraints Exploration can be constrained for selected positions:

```
<Space_Constraint>
489  LYS TRP
490  ASN ARG{1,8,12}
</Space_Constraint>
```

limits two positions to certain types and, in one case, rotamers (numbered as in the “backbone” file).
Reference or unfolded energies  They are essential for many applications. They can be specified for all or selected positions:

```xml
<Ref_Ener>
  ARG  -42.51
  ASN  -11.76
</Ref_Ener>
```

or, using existing labels:

```xml
<Ref_Ener>
  CYS exposed  -1.09
  CYS buried    2.60
  TYR exposed  -4.34
  TYR buried   -1.92
</Ref_Ener>
```

Restarting from a given state  One sometimes needs to start a simulation from a specific state, such as the endpoint state of a previous trajectory, either included in the command file:

```xml
<Seq_Input>
  4 15 1 214 204 2 9 0
</Seq_Input>
```

or copied to a restart.seq file with a single line and read:

```xml
<Seq_Input_File>
  restart.seq
</Seq_Input_File>
```

Reread a trajectory and recompute energies  One sometimes needs to reread a simulation trajectory and recompute energies, possibly using a modified energy function. For this, one reads a trajectory or list of states, say trajectory.seq, using the mode MC. The effect is to launch an MC simulation for each state in the file, which should be done with care. The trick is to set the MC step number to zero (sic). In that case, each MC “trajectory” will simply recompute the energy of its starting state:

```xml
<Mode>
  MONTECARLO
</Mode>
```
5.3. SELECTED OPTIONS FOR MONTE CARLO EXPLORATION

<Seq_Input_File>
trajectory.seq
<\Seq_Input_File>

<Trajectory_Length>
0
<\Trajectory_Length>

Definition of groups  Groups can be defined for several purposes, such as energy function weighting:

<Group_Definition>
pept 1-9  # residues 1-9 are a peptide
prot 134-190  # residues 134-190 are a protein
</Group_Definition>

Selected positions can also be labelled:

<Label>
exposed  11 15 17 19  # exposed residues
buried   12 13 14 16 18 20 21  # buried residues
</Label>
Chapter 6

Multi-backbone Monte Carlo

Multi-backbone MC is described in a recent article [17]. A tutorial and detailed documentation are underway. The idea is to use a model where all or part of the protein backbone can have several discrete conformations. For example, a flexible loop in a binding site might be allowed to occupy a dozen distinct conformations. The conformations could be produced ahead of time by running a short MD simulation using GB solvent, with protX or another tool. During the MC simulation, the backbone conformations will be explored at the same time as the sequence and rotamer spaces. The exploration algorithm is a so-called hybrid MC method, which samples sequences, rotamers, and backbones rigorously according to a Boltzmann distribution. The method requires an assumption about the relative energies of the backbone conformations. Typically, one might assume that conformations sampled by room temperature MD have the same energy [17].

Proteus is currently the only CPD tool that performs multi-backbone CPD while sampling according to a Boltzmann distribution. Having the physically correct distribution makes it possible to obtain rigorous thermodynamic properties such as binding constants or acid/base constants. To achieve Boltzmann sampling, a hybrid MD scheme is used, where a trial backbone change is followed by a short series of MC steps (around 50) where only rotamers can change. At the end of the relaxation period, an acceptance test is applied, where the acceptance probability is obtained as a sum over the relaxation steps (a path integral). For more details, see the original paper [17].
Part III

Selected tasks
Chapter 7

Installation and testing

The Proteus distribution contains a static executable file for protX and one for protMC, which should run on recent Intel processors. Therefore, no compilation is necessary. In addition, a Makefile is provided that can build the executable files using the Intel Fortran and C compilers. Finally, the rest of Proteus is based on bash, python and perl scripts. These will run under ordinary Linux distributions, which include perl and python by default. Compilation can be done using multiple cores in parallel. Using a recent, 16-core machine, compilation of the entire package takes a few seconds.

The tutorials can be run to test the distribution. In addition, the protX directories include a test directory containing around 40 protX scripts that can be run individually to make sure all the features of protX are correctly in place.
8.1 Editing the energy matrix

An important advantage of the precomputed energy matrix is that it can be edited to change selected parameters, with no extra computational effort. Thus, a matrix can be computed with a given dielectric constant $\epsilon_p$ for the protein, or a given set of surface coefficients for the SA energy term. Then the matrix can be edited to use a different $\epsilon_p$ or different coefficients, instead of recomputing a matrix. A perl script is provided, modify_matrix.pl.

8.2 Making Gly active

In many applications, there is a Gly residue in the wildtype system that one would like to mutate as part of the design. Let $N$ be the corresponding residue number. To make position $N$ active with Proteus, there are two key steps. The first is to perform the system build with a modified sequence, where Ala replaces the wildtype Gly$_N$. The Ala$_N$ side chain $C_{\beta}$ can be positioned with any modeling tool, including protX (choose a rough first guess, construct the methyl hydrogens with hbuild, then do some restrained energy minimization) or Scwrl (Dunbrack et al), leading to an appropriate model.pdb file. It is not a problem if the Ala$_N$ side chain overlaps with another, nearby side chain, say position $M$, as long as position $M$ can be mutated to a smaller type as part of the design (ie, it should also be active). Now that a new “wildtype” system has been built, one can set position $N$ to be active in select.str, run the setup steps, and compute the energy matrix, as usual. At this point, position $N$ is set to be active, but Gly is not be part of its mutation space. The second key step is to add Gly to the mutation space, by editing the matrix.bb file to add Gly at position $N$. Notice that no off-diagonal matrix elements are needed, since Gly has no side chain. An example of the relevant diagonal matrix elements is shown below:
8 GLY G 1 2.97 0.00 0.00 0.00

Notice that for Gly at position N, some care must be taken when choosing the corresponding unfolded energy, or $E_{uf}$. Normally, these are computed from a simple tripeptide model (see ligand binding tutorial, part I, above). For Gly, another contribution is necessary, which reflects the more favorable unfolded state entropy for this residue (reflected by its expanded Ramachandran plot). We estimate this effect from the experimental α-helix propensity difference between Ala and Gly, about 1 kcal/mol in favor of unfolded Gly. In practice, we suggest subtracting 1 kcal/mol from the Gly $E_{uf}$, on top of the tripeptide estimate, making Gly harder to insert in the folded protein.

### 8.3 Rotamer library organization

We end this chapter by describing the rotamer organization in Proteus. The protein rotamer libraries are stored in $CPD/rotamer$. The library recommended for use with the Amber ff99SB force field is in $CPD/rotamer/ff99SB/Tuffery95_bbnd_H$. 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 protX stream files. For example, the files corresponding to the serine (SER) sidechain are:

<table>
<thead>
<tr>
<th>directory</th>
<th>files for SER side chains content</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rota</strong></td>
<td>SER_1.pdb,..., SER_9.pdb</td>
<td>3D coordinates for each rotamer</td>
</tr>
<tr>
<td><strong>Chis</strong></td>
<td>SER_1.dat,..., SER_9.dat</td>
<td>Torsion angle values</td>
</tr>
<tr>
<td><strong>Nbrot</strong></td>
<td>SER.dat</td>
<td>Number of rotamers</td>
</tr>
<tr>
<td><strong>Pick</strong></td>
<td>SER.dat</td>
<td>Stream file to extract the side chain torsion values for a current 3D structure</td>
</tr>
<tr>
<td><strong>Rest</strong></td>
<td>SER.dat</td>
<td>Stream file to apply dihedral restraints corresponding to a current rotamer</td>
</tr>
</tbody>
</table>

Specifically, the files look like this:

**Chis/SER_1.dat:**

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

**Nbrot/SER.dat:**

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

**Pick/SER.inp:**


8.4. USING NATIVE ROTAMERS

All that is needed is to activate a flag in parameters.str.
Chapter 9

Optimizing unfolded state energies

9.1 Overview

This chapter describes a tutorial, provided with the Proteus distribution, that shows how to obtain an empirically-optimized model of the unfolded state. Files are in the test directory `opti_eref/`. Although optimizing the unfolded model is not relevant for ligand binding applications, it is very important for whole-protein design. The unfolded state plays a key role because in Proteus, whenever a mutation is attempted in the folded structure, it is accompanied by the reverse mutation in the unfolded structure (Fig. 5.2). The unfolded state is not described by a detailed structural model. Instead, the unfolded energy is a sum of independent contributions from all residues, which depend on the side chain type but not the 3D structure. Let \( E_{uf}^i \) be the contribution of residue \( i \). This contribution depends on the side chain type \( t_i \) at position \( i \): \( E_{uf}^i = E_{uf}^{t_i}(t_i) \). Often, we assume there is no dependency on the residue number: \( E_{uf}^{t_i}(t_i) = E_{uf}(t_i) \). With this assumption, the unfolded state is characterized by a set of “unfolded energies” \( E_{uf}(t) \) that depend on the side chain type \( t \) but not its position in the polypeptide chain. The values \( E_{uf}(t) \) are normally chosen so that a simulation will reproduce the overall amino acid composition of a set of natural homologs. This amounts to maximizing the probability or likelihood of sampling the natural sequences during the design. The underlying theory is described elsewhere [18], while the practical procedure is described below. The procedure incrementally adjusts a set of \( E_{uf}(t) \) values, following the direction of the likelihood gradient, and stopping when further adjustments do not increase the likelihood; i.e., the maximum-likelihood values have been reached. We recall that the gradient of the log-likelihood has the form:

\[
\frac{1}{N} \frac{\partial}{\partial E_{uf}(t)} \ln L = \frac{1}{N} \sum_s n_s(t) - \langle n(t) \rangle = \frac{N(t)}{N} - \langle n(t) \rangle \tag{9.1}
\]
Here, $N$ is the number of amino acids in the target database, $N(t)$ is the number with type $t$, $n(t)$ is the number seen in the MC simulation, and the brackets represent an average over the simulation \[18\]. Thus, to maximize $\mathcal{L}$, we should choose $E_{\text{uf}}(t)$ such that a long simulation gives the same amino acid frequencies as the target database: $\frac{N(t)}{N} = \langle n(t) \rangle$ for all types $t$.

### 9.2 Practical procedure

We assume we are optimizing unfolded energies for a particular protein family, represented by $n = 4$ proteins. In the tutorial, these are PDZ proteins with the PDB codes 1G9O (NHERF), 2BYG (DLG2), 1KWA (Cask) and 1N7E (Grip). We will refer to them as proteins A–D. For each one, we assume the build and setup have been done and the energy matrix computed. In the tutorial, each protein has its own directory, with the usual subdirectories build, matrix, and so on.

An important ingredient is the target amino acid composition. This is obtained from a sequence alignment, created ahead of time by the user, and which includes proteins A, B, C, D. The composition is then obtained using a perl script (provided in the tutorial). In the tutorial, we apply an unfolded state model where amino acid positions are grouped according to their buried or exposed character in the folded protein. Each group will have its own set of type-dependent unfolded energies. This model assumes that the folded structure affects the unfolded model, either because residual folded structure is retained in the unfolded state, or because the folded model compensates for some of the errors in the unfolded model \[18\]. The perl script that computes the amino acid composition uses residue burial information from the 3D structures to distinguish between the buried and exposed compositions.

Given the amino acid compositions, the idea is to start from an initial set of unfolded energies, run a set of MC simulations for each protein (2–3 per protein), and compare the computed composition to the target one. The unfolded energies are then updated, by adding an increment that is related to the likelihood gradient. The procedure is repeated: a new set of MC simulations is done, the composition computed, and the unfolded energies updated. The procedure is run until convergence. In the tutorial, the $E_{\text{uf}}(t)$ update rule is the “linear” rule \[18\]:

\[
E_{\text{uf}}(i+1) = E_{\text{uf}}(i) + \alpha \frac{\partial}{\partial E_{\text{uf}}} \ln \mathcal{L} = E_{\text{uf}}(i) + \delta E \left( n_{\text{exp}}(t) - \langle n(t) \rangle_{n} \right)
\]

Here, $i$ is an iteration number; $\alpha$ is a constant; $n_{\text{exp}} = N(t)/N$ is the mean population of amino acid type $t$ in the target database; $\langle \rangle_{n}$ indicates an average over a simulation done using the current unfolded energies $\{E_{\text{uf}}(n)\}$, and $\delta E$ is an empirical constant with the dimension of an energy, referred to as the update amplitude. We have
omitted the distinction between buried and exposed positions here for simplicity.

In the tutorial, in each MC simulation, every other amino acid position is active, while the rest are inactive. Thus, there are two sets of active positions (per protein), and one simulation is done for each set. The overall computed amino acid composition is thus obtained at each iteration from eight simulations, two per protein.

In the tutorial, the procedure is done using a single computer (localhost), defined in a file (./project.info). This file can be edited to use several computers. The computers should all have access to a shared disk (mounted through NFS), where all the data are stored. Each MC simulation uses REMC with eight replicas; two simulations are run at a time, corresponding to one of the four proteins. Changes to this setup can be made in a fairly straightforward way; for example, one might want to do REMC with 4 replicas and run 3 simulations per protein or per computer. Recall that the procedure starts with the energy matrices already in place.

### 9.3 Running the tutorial

During the tutorial, we will:

- Compute the target frequencies from a sequence alignment
- Compute initial guesses for the unfolded energies
- Optimize the unfolded energies iteratively

<table>
<thead>
<tr>
<th>Table 9.1: Main directories and files</th>
</tr>
</thead>
<tbody>
<tr>
<td>1G9O/</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>./</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>lib/</td>
</tr>
<tr>
<td>src/</td>
</tr>
</tbody>
</table>

We suppose a sequence alignment of proteins A–D and their homologs has been prepared: ./lib/all_seq.aln. Here, we consider four PDZ domains (PDB codes: 1G9O, 1KWA, 1N7E, 2BYG). For each one, the energy matrices are available (.1G9O/matrix.bb|pw and so on). In addition, the PDB structures from each setup
CHAPTER 9. OPTIMIZING UNFOLDED STATE ENERGIES

step (initially stored in matrix/setup_nogiant.pdb) have been copied and renamed to ./1G9O/setup.pdb, and so on. A key file is ./project.info, where the proteins are defined, as well as the computers to use. This should be edited for each new project. This file will be read by several bash scripts.

In the unfolded model, we distinguish buried and exposed positions, with distinct sets of unfolded energies and target amino acid frequencies. Exposed/buried positions are identified using accessible surface information computed during the setup step and contained in the setup.pdb files. Most positions have the same buried or exposed character in all four proteins A–D. If not, the buried or exposed character is averaged over the four proteins. Thus, if a position \(i\) is buried in A–C but exposed in D, it will contribute to the buried frequencies with a weight of 0.75 and to the exposed frequencies with a weight of 0.25.

To compute the frequencies from the sequence alignment (with the help of the setup.pdb info), run the command:

```bash
./init_f.sh
```

Outputs:

- `exp_buried.freq`: buried frequencies
- `exp_exposed.freq`: exposed frequencies
- `1G9O/setupN.pdb`: position information

| Table 9.2: Experimental frequencies (%) obtained from the sequence alignment |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
|                             | exposed | buried | exposed | buried | exposed | buried |
| ALA                         | 3.99    | 6.77   | 1.40    | 3.90   |
| CYS                         | 0.41    | 1.53   | 4.02    | 3.59   |
| THR                         | 5.45    | 5.88   | 5.23    | 4.25   |
| GLU                         | 9.49    | 5.21   | 6.53    | 3.08   |
| ASP                         | 3.93    | 4.11   | 7.91    | 3.79   |
| PHE                         | 1.30    | 2.59   | 0.96    | 1.32   |
| TRP                         | 0.02    | 0.02   | 0.00    | 0.00   |
| ILE                         | 3.31    | 14.54  | 0.00    | 0.00   |
| VAL                         | 5.74    | 12.86  | 5.13    | 2.02   |
| LEU                         | 4.47    | 15.57  | 4.71    | 1.79   |
| LYS                         | 8.31    | 4.53   | 17.68   | 2.65   |

The next step is to obtain initial guesses for the unfolded energies. A reasonable starting guess will lead to faster convergence. We use the matrix diagonal files
(matrix.bb) to obtain the initial guess. For each protein, position, and amino acid type, we take the minimum energy rotamer, and average its energy over types, positions, and proteins. To execute this procedure, Step 3, run the command:

```bash
./init_e.sh
```

Outputs:

- **buried_0.ener**: buried frequencies
- **exposed_0.ener**: exposed frequencies

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Exposed</th>
<th>Buried</th>
<th>Exposed</th>
<th>Buried</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALA</td>
<td>3.62</td>
<td>4.93</td>
<td>-9.68</td>
<td>-3.80</td>
</tr>
<tr>
<td>CYS</td>
<td>3.45</td>
<td>5.93</td>
<td>-1.12</td>
<td>4.35</td>
</tr>
<tr>
<td>THR</td>
<td>-7.08</td>
<td>-4.52</td>
<td>-24.72</td>
<td>-23.00</td>
</tr>
<tr>
<td>ASP</td>
<td>-18.63</td>
<td>-15.43</td>
<td>-3.55</td>
<td>-2.04</td>
</tr>
<tr>
<td>PHE</td>
<td>12.55</td>
<td>15.17</td>
<td>-55.94</td>
<td>-54.55</td>
</tr>
<tr>
<td>TRP</td>
<td>11.52</td>
<td>16.44</td>
<td>-0.96</td>
<td>1.85</td>
</tr>
<tr>
<td>ILE</td>
<td>1.96</td>
<td>6.91</td>
<td>-6.43</td>
<td>-0.59</td>
</tr>
<tr>
<td>VAL</td>
<td>-0.93</td>
<td>2.33</td>
<td>-6.94</td>
<td>-0.80</td>
</tr>
<tr>
<td>LEU</td>
<td>-4.28</td>
<td>0.38</td>
<td>-4.01</td>
<td>2.29</td>
</tr>
</tbody>
</table>

With the target amino acid frequencies and the initial guess for the unfolded energies in place, we can prepare the directories for the MC simulations that will be performed. For each protein, we consider two batches of residues, and perform simulations where one batch is allowed to mutate (active batch) while the other does not mutate (inactive batch). The batches are formed by taking every other amino acid in the sequence. Thus, at each iteration of the optimization process, there will be two simulations per protein. The association between computers, proteins, and batches, is set up in the project.info file:

```
# computer    local directory        protein     batch
localhost/home/dupont/test_case/opti_eref/  1G9O  1
localhost/home/dupont/test_case/opti_eref/  1G9O  2
localhost/home/dupont/test_case/opti_eref/  1N7E  1
localhost/home/dupont/test_case/opti_eref/  1N7E  2
localhost/home/dupont/test_case/opti_eref/  1N7E  1
localhost/home/dupont/test_case/opti_eref/  1N7E  2
```
For a new project, since the MC calculations are rather demanding, one may want to edit project.info to use multiple computers. The next step, **Step 4**, is to prepare working directories (locally or remotely), by running the command:

```bash
./init_m.sh
```

For the MC simulations, we use REMC with 8 replicas and the parameters:

- Trajectory length: 500 K MC steps
- GB method: FDB
- Temperatures ($kT$): 3; 2; 1.333; 0.888; 0.592; 0.395; 0.263; 0.175

Finally, we run **Step 5** the main, iterative optimization: `.iteration.sh`. Here, only two iterations are run; `iteration.sh` can be edited to run a greater number. At the end of each iteration, the amino acid frequencies of the produced sequences are computed. Specifically, the frequencies are computed from the MC simulation of replica 6, which has a temperature of $kT = 0.263$ kcal/mol (about half of room temperature). Once the frequencies are computed for each protein and batch of residues, they are averaged and compared to the target frequencies. The unfolded energies are then updated using the “linear” rule.

**The main output files are:**

- `ref_ener/buried_<...>.ener`: buried unfolded energies
- `ref_ener/exposed_<...>.ener`: exposed unfolded energies
- `frequencies/buried_<...>.freq`: buried frequencies
- `frequencies/exposed_<...>.freq`: buried frequencies

Finally, one can use the produced unfolded energies in a standard exploration protMC simulation, with the help of the **Label** and **Ref_Ener** protMC tags:

```xml
<Label>
exposed 9 10 ...
buried 15 17 ...
</Label>
<Ref_Ener>
ALA exposed 3.496
ALA buried 4.799
CYS exposed 3.282
CYS buried 5.768
```
... 

</Ref_Ener>
Chapter 10

Adding D-amino acids at a specific position

Suppose we want to allow D-amino acids at position $N$ in a protein. This might occur in a binding site design, say, where a wildtype Gly can be advantageously augmented by a long D side chain that points towards the desired ligand (whereas an L sidechain points in the wrong direction). Although the method to do this is simple, Proteus does not provide complete automation for the moment. Therefore, D-amino acids should only be used once one has some familiarity with Proteus. Nevertheless, we chose to include the information below, even if it is only practical for experienced users.

The main ingredient is a rotamer library that contains the D side chain conformers. Such a library can be homemade and will be provided with a future Proteus distribution. It is built by applying a mirror reflection to the usual L rotamers. While this should be a good approximation, some refinement of the D library may be desirable in the future. The design proceeds as follows:

1. Run the build step, replacing the wildtype Gly$_N$ with D-Ala by putting the D methyl group in a sensible position (model.pdb file).

2. Perform the setup.sh step as usual, in the matrix directory.

3. Perform the runI.sh step as usual, with position $N$ inactive (specified in sele.str).

4. Perform the runI.sh step again, in another matrix directory, with only $N$ active, using the D rotamer library for the whole system. From this step, we obtain the correct positions of the D rotamers of residue $N$. They are found in the local rotamer directory: matrixTMP/local/Rota, say. Everything else produced by this step will be discarded.
5. Go back to the *initial matrix directory*, replace the rotamers of residue $N$ by the D rotamers just obtained (i.e., copy the residue $N$ rotamers from one matrix/local/Rota directory to the other). Notice that the number of D rotamers is the same as the number in the usual, L rotamer library.

6. Rerun the matrixI.inp step of the runI.sh procedure (but NOT the setupI.inp step, which should be commented out). From now on, as far as Proteus knows, position $N$ can only have D-amino acids.

7. Continue with the usual steps: runJ.sh to obtain the rest of the matrix, followed by MC exploration with protMC.
Chapter 11

Using Toulbar2 for exact optimization

The Toulbar2 program, developed by Thomas Schiex and coworkers, provides several exact and heuristic algorithms to optimize sequences and rotamers and identify the Global Minimum Energy Conformation or GMEC [19, 20]. The Sdc1 tutorial found at $CPD/tutorials/tuto_Sdc1 includes a section (subdirectory toulbar) that shows how to use Toulbar2 with a Proteus energy matrix [13, 21]. We provide a bash and a perl script to convert the Proteus energy matrix to a format that can be understood by Toulbar2. Specifically, the matrix is converted to a set of positive integers by scaling and shifting the energies, without changing the physical nature of the matrix. Toulbar2 can then be run using a bash script, such as the one provided. The Toulbar2 program itself should be obtained directly from its authors (see eg, https://github.com/toulbar2/toulbar2, http://www7.inra.fr/mia/T/toulbar2/).
Part IV

Solvent models in Proteus
Chapter 12

Surface area calculations

The main types of surfaces are defined in Fig. 12.1: van der Waals surface, solvent accessible surface, and molecular or “Connolly” surface. In protX, mainly use the solvent accessible surface, or solvent accessible surface area, or SASA. The “contact” area is still another surface, available but rarely used. Two approximate methods for SASAs are also provided.

12.1 Accessible Surface Area in protX

The algorithm by Lee and Richards (1971) is used to compute the solvent accessible surface area (MODE=ACCEss) or the contact area (MODE=CONTact). The routine uses the van der Waals radii (Eq. 18.6) for all atoms as specified in the parameter statement. Upon completion, the accessible surface area for each selected atom is stored in the RMSD atom property (in Å² units).
12.1.1 Syntax

\texttt{SURFace} \{ <\texttt{surface-statement}> \} \texttt{END} is invoked from the main level of protX. The \texttt{END} statement activates execution.

\texttt{<surface-statement>:=}

\begin{itemize}
  \item \texttt{ACCUracy=<real>} is accuracy of the numerical integration (default: 0.05).
  \item \texttt{MODE=ACCESS | CONTact | FFVG | LCPO} is access or contact mode, or an approximate ASA method, FFVG or LCPO (default: access).
  \item \texttt{RH2O=<real>} is probe radius (default: 1.6 Å).
  \item \texttt{SELEction=<selection>} performs the calculation for the selected atoms.
\end{itemize}

12.1.2 Example

Here the accessible surface area is computed and printed.

\begin{verbatim}
surface
  rh2o=1.6
  mode=access
end
vector show elem ( rmsd ) ( not hydrogen )
\end{verbatim}

12.2 Approximate Fraternali or FFVG method

12.2.1 Definitions

An approximate ASA calculation was proposed by Fraternali and van Gunsteren \cite{22}. The ASA $A_i$ of an atom $i$ is defined by the approximate analytical expression \cite{23}:

\begin{equation}
A_i = S_i \prod_j (1 - p_{ij} b_{ij}(r_{ij})/S_i)
\end{equation}  \hspace{1cm} (12.1)

Here, $S_i$ is the ASA of an isolated atom of radius $R_i$, assuming a solvent probe radius of $R_w$:

\begin{equation}
S_i = 4\pi(R_i + R_w)^2
\end{equation}  \hspace{1cm} (12.2)

ASA removed by another atom $j$ is given by:

\begin{align}
b_{ij} &= 0 \quad r_{ij} > R_i + R_j + 2R_w \\
&= \pi(R_i + R_w)(R_i + R_j + 2R_w - r_{ij}) (1 + (R_j - R_i)/r_{ij}) \\
r_{ij} < R_i + R_j + 2R_w
\end{align}  \hspace{1cm} (12.3)
12.2. APPROXIMATE FRATERNALI OR FFVG METHOD

The parameter $p_i$ depends on the atom type and reduces double counting when atom $i$ overlaps with several other atoms. The parameter $p_{ij}$ distinguishes between first and second covalent neighbors of atom $i$. If atom $j$ is covalently bound to $i$, then $p_{ij} = 0.8875$. If atoms $i$ and $j$ are both bound to another atom (they form a covalent angle), then $p_{ij} = 0.3516$. Otherwise, $p_{ij} = 0.3156$. Values of $p_i$ that are compatible with the Amber ff99SB force field are listed in Table 12.1. Further optimization of these values is needed.

12.2.2 Implementation in protX

The FFVG method is included as an option of the surf command. The atomic $p_i$ parameters can be read in from a file:

```latex
@surf_FFVG.str ! read the $p_i$ parameters ! Compute surface area with mode FFVG surf rh2o=1.50 accu=0.01 selec=(resid 9) mode=FFVG FACBond=0.8875 FACTheta=0.3516 FACDefault=0.3156 end vector show (rmsd) (resid 9) ! display ASA for residue 9 atoms
```

The $p_i$ parameter file looks like this:

```latex
! type $p_i$
param surf C  1.554 end
param surf CA  1.073 end
param surf CC  1.554 end
param surf CR  1.073 end
param surf CT  1.554 end
param surf CV  1.554 end
param surf H   1.128 end
param surf H1  1.128 end
param surf H4  1.128 end
param surf H5  1.128 end
param surf HA  1.128 end
param surf HC  1.128 end
param surf HO  0.944 end
param surf HP  1.128 end
param surf HS  0.928 end
param surf N   1.028 end
param surf N2  1.028 end
param surf N3  1.028 end
param surf NA  1.028 end
```
In other words, the $p_i$ are read by the option surf of the param command. The parameters are provided in the directory $CPD/lib$. For now, FFVG is not implemented as an energy term in protX; this will be done soon.

<table>
<thead>
<tr>
<th>GROMOS</th>
<th>Amber</th>
</tr>
</thead>
<tbody>
<tr>
<td>atom type</td>
<td>$p_i$</td>
</tr>
<tr>
<td>OA, OW</td>
<td>1.080</td>
</tr>
<tr>
<td>O</td>
<td>0.926</td>
</tr>
<tr>
<td>OM</td>
<td>0.922</td>
</tr>
<tr>
<td>NT, NL</td>
<td>1.215</td>
</tr>
<tr>
<td>N</td>
<td>1.028</td>
</tr>
<tr>
<td>NR5, NR5*</td>
<td>1.028</td>
</tr>
<tr>
<td>NR6, NR6*</td>
<td>1.028</td>
</tr>
<tr>
<td>NZ</td>
<td>1.028</td>
</tr>
<tr>
<td>NE</td>
<td>1.028</td>
</tr>
<tr>
<td>C, CB</td>
<td>1.554</td>
</tr>
<tr>
<td>CH1</td>
<td>1.276</td>
</tr>
<tr>
<td>CH2</td>
<td>1.045</td>
</tr>
<tr>
<td>CH3</td>
<td>0.880</td>
</tr>
<tr>
<td>CR51</td>
<td>1.073</td>
</tr>
<tr>
<td>CR61</td>
<td></td>
</tr>
<tr>
<td>HO, HW</td>
<td>0.944</td>
</tr>
<tr>
<td>H</td>
<td>1.128</td>
</tr>
<tr>
<td>HS</td>
<td>0.928</td>
</tr>
<tr>
<td>S</td>
<td>1.121</td>
</tr>
</tbody>
</table>
12.3 Approximate LCPO method

The LCPO method (Linear Combination of Pairwise Overlaps) \cite{24} is similar to Fraternali, above. The ASA of an atom \( i \) is approximated as

\[
A_i = S_1 S_1 + P_2 \sum_{j \in N(i)} A_{ij} + P_3 \sum_{j \in N(i)} \sum_{k \in N(j), k \neq j} A_{jk} + P_4 \sum_{j \in N(i)} \left( \sum_{j \in N(i)} \sum_{k \in N(j), k \neq j} A_{jk} \right)
\]  

(12.5)

\( N(i) \) designates the set of atoms that overlap with atom \( i \) (its “neighborhood”). \( P_1 - P_4 \) are empirical parameters. \( A_{ij} \) is the area of atom \( i \) that is buried by (inside) atom \( j \):

\[
A_{ij} = 2 \pi R_i \left( R_i - \frac{1}{2} r_{ij} - \frac{R_i^2 - R_j^2}{2r_{ij}} \right) \quad r_{ij} < R_i + R_j
\]

(12.6)

\( A_i \) and \( A_{ij} \) are simple functions of atomic coordinates, whose derivatives are readily computed \cite{23}. Empirical values for \( P_1 - P_4 \) were reported by Hasel et al \cite{23}.

12.3.1 Implementation in protX: the ESURF energy term

The LCPO method is implemented as an option for the surf command:

```
surf rh2o=1.50 accu=0.01 mode=LCPO selec=(resid 9) end
```

The type-dependent parameters \( P_1 - P_4 \) are defined in the file surf_LCPO.str in the directory \$CPD/lib.

LCPO will soon be implemented as an energy term, \$ESURF, activated by the usual flags statement:

```
flags include surf end
```

Implementation of the exact and FFWG variants as energy terms will be available soon. The surf term contributes both to the energy and forces, as with other energy terms.
Chapter 13

Nonpolar solvation

13.1 Theory

The solute-solvent interaction consists of an electrostatic part where the atomic charges in the low dielectric cavity (solute) interact with the high dielectric surrounding medium (solvent) as described by the Generalized Born (GB) model, and a nonpolar part which describes the cavity formation and the van der Waals solute-solvent dispersion interaction.

13.1.1 Solute-solvent van der Waals dispersion model

In the spirit of the Weeks-Chandler-Andersen (WCA) repulsive/attractive decomposition of the nonpolar contribution to the solvation free energy [25], we model the solute-solvent van der Waals dispersion interactions using the attractive part of the Lennard-Jones potential. Following the continuum solute-solvent van der Waals (vdW) energy model of Gallicchio et al. [26], the average vdW dispersion interaction of atom $i$ with water is given by the integral of the attractive LJ potential between atom $i$ and the oxygen atom of the water molecule, over the solvent volume, where the water number density $\rho_w$ is assumed constant:

$$\Delta G^{DI} = \sum_i U_{i,vdW}^{vdW}$$ (13.1)

$$U_{i,vdW} = -\rho_w \int_{\text{solv}} \frac{4\epsilon_{iw}\sigma_{iw}^6}{|r - r_i|^6} d^3r$$ (13.2)

$\epsilon_{iw}$ and $\sigma_{iw}$ are the LJ potential parameters for the solute atom-water oxygen pair. The total solute-solvent vdW dispersion interaction is given by the sum of the individual vdW interactions of all atoms of the solute. The integral in the above equation can be re-written as the difference of the integral over the whole space and the solute region outside the vdW radius $R_i$. In other words, the solvation
free energy of an isolated atom $i$ fully solvated, is reduced by the presence of all
surrounding solute atoms $j$:

$$\bar{U}_{vdW}^i = -\frac{4}{3}\epsilon_{ij}\sigma_{ij}^6\left(\frac{4\pi}{3R_i^3} - \int_{r>R_i} \frac{1}{|r - r_i|^6} d^3r\right)$$  \hspace{1cm} (13.3)

$$\bar{U}_{vdW}^i = -\frac{f_i}{R_i^3} + f_i\left(\frac{3}{4\pi} \int_{r>R_i} \frac{1}{|r - r_i|^6} d^3r\right), \quad f_i = \frac{16\pi}{3}\epsilon_{ij}\sigma_{ij}^6$$  \hspace{1cm} (13.4)

The integral of $1/r^6$ over the solute region is approximated by two contributions
as proposed by Onufriev [27] and computed analytically. The main integral $I_{vdW}^i$ is
over the atomic vdW spheres and the correction integral $I_{neck}^i$ is over the “neck”-
shaped free space regions between pairs of vdW spheres:

$$\frac{3}{4\pi} \int_{r>R_i} \frac{1}{|r - r_i|^6} d^3r \approx I_{vdW}^i + I_{neck}^i$$  \hspace{1cm} (13.5)

$$I_{vdW}^i = \frac{3}{4\pi} \int_{r>R_i} \frac{1}{|r - r_i|^6} d^3r = \sum_{j\neq i} I_{vdw}^{ij}(r_{ij}, R_i, S_{vdw} R_j)$$

$$I_{neck}^i = \frac{3}{4\pi} \int_{r>R_i} \frac{1}{|r - r_i|^6} d^3r = \frac{3}{4\pi} \sum_{j\neq i} I_{neck}^{ij}(r_{ij}, R_i, R_j)$$  \hspace{1cm} (13.6)

Both terms in Eq. (13.5) are computed by the sum $\sum_{j\neq i} I_{ij}$ of all atom pairwise
interactions expressed by the following analytical conditional functions:

$$I_{vdW}^{ij}(R_i, R_j, r_{ij}) = \begin{cases} \frac{R_i^3}{(r_{ij}-R_i^3)^3}, & \text{if } r_{ij} \geq R_i + R_j \\ \frac{1}{16r_{ij}} \left(\frac{r_{ij} + 3R_i}{r_{ij} + R_i} \right)^3 + \frac{3(R_i^2-R_j^2-(r_{ij}-R_i)^2)+2r_{ij}R_i}{R_i^4}, & \text{otherwise} \end{cases}$$  \hspace{1cm} (13.7)

$$I_{neck}^{ij}(R_i, R_j, r_{ij}) = \begin{cases} A_{ij}(r_{ij} - B_{ij})^4(R_i + R_j + 2R_w - r_{ij})^4, & \text{if } B_{ij} < r_{ij} < R_i + R_j + 2R_w \\ 0, & \text{otherwise} \end{cases}$$  \hspace{1cm} (13.8)

The neck parameters $A$ and $B$ themselves depend on the atomic vdW radii of each
pair and the water probe radius $R_w$. Finally, the total solute-solvent vdW dispersion
interaction is given by

$$\Delta G^{DI} = \sum_i -\frac{f_i}{R_i^3} + \sum_i \sum_{j \neq i} f_i(I_{vdw}^{ij} + \frac{3}{4\pi} S_{neck} I_{neck}^{ij})$$  \hspace{1cm} (13.9)

### 13.1.2 Gaussian Nonpolar Solvent Model

The Lazaridis-Karplus (LK) model [28] (also referred to as EEF1) expresses the
total solvation free energy of a particular molecular conformation as a sum over
13.2 Implementation

The solute-solvent vdW dispersion interaction was coded at the level of the non bonded Generalized Born (GB) evaluation as an individual subroutine. After an energy call, the parameters of the model described above are read and stored. That is, type and atom-based van der Waals radii ($R_{vdw}$), the solvent type (oxygen atom for water), and parameters $A$, $B$ if the neck correction term is turned on. In a first stage, the van der Waals interaction of each isolated atom of the solute with the continuum solvent is accumulated to give the “self” part of the dispersion energy. The factor in the numerator of Eq. (13.4) depends on the water number density and the Lennard-Jones B coefficient for each solute-solvent atom pair. In a second
stage, the interactions of all solute atoms \( j \) surrounding each atom \( i \) are evaluated and summed, to give the “interaction” part, which accounts for the replacement of surrounding solvent by solute. As shown in Eq. (13.6), the van der Waals radii of the surrounding atoms are scaled down by a factor \( S_{vdw} \). The neck term corrects the simplistic representation of the solute as a set of van der Waals spheres, and accounts for omitted space between vDW spheres within the solute. The total neck contribution is scaled by a factor \( S_{neck} \) (Eq. 13.6). The parameters \( A \) and \( B \) in Eq. (13.8) depend on the pair of \( R_{vdw}^i, R_{vdw}^j \) and the water probe radius \( R_w \). A set of \( A \) and \( B \) values has been evaluated on a 2-dimensional equally-spaced grid \((R_{vdw}^i \times R_{vdw}^j)\) using the numerical method NSR6 developed by Onufriev et al. [29]. If the van der Waals radii of a pair do not coincide with a grid node, then \( A \) and \( B \) parameters are obtained by cubic spline interpolation. The second derivative values of neck parameters are also read from the parameter file.

The solute-solvent van der Waals dispersion interaction term is pairwise decomposable, the derivative has an analytical form and the forces are readily obtained:

\[
\frac{d \Delta G^{PI}}{d r_{ij}} = \begin{cases} 
\sum_{j \neq i} \frac{-6r_{ij}^3}{(r_{ij}^2 - R_i^2)^3}, & \text{if } r_{ij} \geq R_i + R_j \\
\sum_{j \neq i} \left( \frac{2}{16r_{ij}} \left( \frac{r_{ij} + 4R_j}{r_{ij} + R_j} \right)^{4} + \frac{3r_{ij} - 4R_i}{R_i^4} \right) - \frac{1}{16r_{ij}^3} r_{vdW}^j (r_{ij} < R_i + R_j), & \text{otherwise}
\end{cases}
\]

(13.13)

The Lazaridis-Karplus gaussian nonpolar solvent model is coded within the non-bonding interaction calculation as a separate subroutine. At first, the type-based parameters of the model, \( G_{ref}, G_{free} \) and \( \lambda \) are read and stored in arrays. A separate, type-based array is needed for use of the model with AMBER, because it was initially developed for CHARMM and there is a poor correspondence between carbon atoms types CT and CH1E/CH2E/CH3E. The nonpolar part of the solvation free energy is computed in two steps. First, we compute the sum of atomic solvation free energies (\( G_{ref} \)) in their isolated state, completely surrounded by solvent (first term in Eq. (13.14)); then we subtract the sum of the desolvation (replacement of solvent by solute) of each atom by all remaining solute atoms \( j \) (second term in Eq. (13.14)). All atom pairs contribute to the LK model, so 1-2 and 1-3 pairs (excluded from the van der Waals and electrostatic interactions) are taken into account. All hydrogen atoms are considered part of the heavy atom they are attached to, and are excluded from the calculation, as in the initial model:

\[
\Delta G^{LK} = \sum_i G_i^{ref} - \sum_i \sum_{j>i} (f_i(r_{ij})V_j + f_j(r_{ij})V_i)
\]

(13.14)

When using the constraint interaction command to compute the interaction energy between two selected groups of atoms, the first term of Eq. (13.14) is evaluated only for those atoms that belong to both selections. If the two groups do not share any
atoms, only the second term is computed, describing the desolvation of the first group by the second and the desolvation of the second by the first. The command `cons inte (resid R1 or resid R2) (resid R1 or resid R2) end` computes the solvation energy of both residues R1 and R2 as follows:

$$
\Delta G^{LK}(R_1, R_2) = \sum_{i \in R_1, R_2} G_i^{ref} - \sum_{i \in R_1, R_2} \sum_{i<j \in R_1, R_2} \left( f_i(r_{ij})V_j + f_j(r_{ij})V_i \right)
$$

(13.15)

The command `cons inte (resid R1) (resid R1 or resid R2) end` computes the solvation energy of residue R1 in the presence of residue R2, correctly taking into account the desolvation of R1 by R2; but it also includes the desolvation of residue R2 by residue R1. To eliminate the latter undesirable contribution we remove $f_j(r_{ij}), j \in R_2$ by setting the atom-based parameters of residue R2 to zero, with the command `parameters GNSP (resid R2) 0.0 0.0 1.0 end`. Now the solvation energy of residue R1 in the presence of R2 is given by

$$
\Delta G^{LK}(R_1, R_1 - R_2) = \sum_{i \in R_1} G_i^{ref} - \sum_{i \in R_1} \sum_{i<j \in R_1, R_2} f_i(r_{ij})V_j
$$

(13.16)

The solvation free energy depends on the distance between pairs of atoms ($r_{ij}$), and its derivative has the analytical form:

$$
\frac{d \Delta G^{LK}}{dr_{ij}} = \left( \frac{1}{r_{ij} - R_i} - \frac{1}{\lambda_i^2} \right) f_i V_j + \left( \frac{1}{r_{ij} - R_j} - \frac{1}{\lambda_j^2} \right) f_j V_i
$$

(13.17)

13.3 Syntax

13.3.1 Solute-solvent van der Waals dispersion energy

The dispersion term is assigned the variable name $GBDI$ and is activated by the flag statement: `flags include GBDI end`

The GBDI options are set as follows:

- **NBONDs**:<nbonds-statement>|<gborn-nbonds-statement>|<gbd-nbonds-statement> END
  - `<gbd-nbonds-statement>`::= GBDI GBDN Flags activating the main GBDI term and the neck contribution. Default: inactive.
  - **WTYPE**=<string> Solvent chemical type. Default: OW (TIP3P oxygen atom).
  - **WRHO**=<real> Solvent density number. Default: 1.
  - **SGBDI**=<real> $R_j^{ndW}$ scaling factor. Default: 1.
  - **SNECK**=<real> Neck term scaling factor. Default: 1.
  - **RWAT**=<real> Water probe radius. Default: 1.4.
13.3.2 Setting up the parameters

The type-based parameters of the vdW dispersion model are set with a parameter statement:

\[
\text{PARAmeter} \{ <\text{parameter-statement}> \} \text{ END}
\]

\[
\text{DSPN} <\text{RvdW-statement}><\text{neckAB-statement}> \text{ END}
\]

\[
<\text{RvdW-statement}> ::= 
\]

\[
\text{GNOD} <\text{integer}> <\text{real}> \ldots <\text{real}> \text{ defines the size of the grid and assigns a vdW radius foreach node of the grid.}
\]

\[
<\text{neckAB-statement}> ::= 
\]

\[
\text{NCKA} <\text{real}> \ldots <\text{real}> \text{ assigns a value of the neck-A parameter to all nodes of a row in the } (R^\text{vdW}_i \times R^\text{vdW}_j) \text{ grid.}
\]

\[
\text{NCKB} <\text{real}> \ldots <\text{real}> \text{ assigns a value of the neck-B parameter to all nodes of a row in the } (R^\text{vdW}_i \times R^\text{vdW}_j) \text{ grid.}
\]

\[
\text{NC2A} <\text{real}> \ldots <\text{real}> \text{ assigns a value of the second derivative of neck-A parameter to all nodes of a row in the } (R^\text{vdW}_i \times R^\text{vdW}_j) \text{ grid, used for cubic spline interpolation.}
\]

\[
\text{NC2B} <\text{real}> \ldots <\text{real}> \text{ assigns a value of the second derivative of neck-B parameter to all nodes of a row in the } (R^\text{vdW}_i \times R^\text{vdW}_j) \text{ grid, used for cubic spline interpolation.}
\]

Lazaridis-Karplus interaction energy

The LK term is assigned the variable name \$GNSM\$ and is activated by the flag statement: flags include GNSM end. The GNSM option is set up as follows:

\[
\text{NBONDs} <\text{nbonds-statement}> | <\text{gborn-nbonds-statement}> | <\text{gbdi-nbonds-statement}> | <\text{gnsm-nbonds-statement}> \text{ END}
\]

\[
<\text{gnsm-nbonds-statement}> ::= \text{GNSM Flag activating the GNSM term. Default: inactive.}
\]

Setting up the parameters The type- and atom-based parameters of the LK model are set with a parameter statement:

\[
\text{PARAmeter} \{ <\text{parameter-statement}> \} \text{ END}
\]

\[
\text{GNSP} <\text{type}> <\text{real}> <\text{real}> <\text{real}> \text{ adds } G^{\text{ref}}, G^{\text{free}} \text{ and } \lambda \text{ parameters for the atom type to the parameter database.}
\]

\[
\text{GNSP} <\text{selection}> <\text{real}> <\text{real}> <\text{real}> \text{ adds } G^{\text{ref}}, G^{\text{free}} \text{ and } \lambda \text{ parameters for the selected atoms to the parameter database.}
\]
13.3.3 Example: minimization and MD with GBDILK

topology
@protX/toppar/amber/masses_parm99.rtf
@protX/toppar/amber/amino_parmparm99SB bbunif.rtf
@protX/toppar/amber/solvents.rtf
@protX/toppar/amber/ions.rtf
end

parameters
@parm99SB.plus.prm {plus DI and LK parameters}
end

structure @allh_model.psf end
coordinates @allh_model.pdb

@LK_charmm2amber.str {type conversion from charmm19 to amber99}

parameter
nbonds
atom trunc cdie eps=1 e14fac=0.8333333333333
ctonnb=97. ctofnb=98. cutnb=99. nbxmod=5 toler=100.
gbhct eps=1. weps=80.
gbdi wtype = OW sgbdi=0.6211 wrho=0.033428 {GBDI parameters}
gbdn sneck=0.4058 rwat=1.4 {GBDN parameters}
gnsm {GNSM option}
end
end
flags include gbse gbin gbdi gnsm end

energy end
minimize powell nstep=50 end
energy end
display $gbse $gbin $gbdi $gnsm

vector do (vx = maxwell(250)) (all)
vector do (vy = maxwell(250)) (all)
vector do (vz = maxwell(250)) (all)
dynamics verlet
nstep=1000  timest=0.001  iasvel=current
nprint=250  iprfrq=250
end

stop
Chapter 14

Generalized Born electrostatics

14.1 Introduction

The Generalized Born (GB) model \[30–33\] 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 \[34–37\]. Two GB variants have been implemented in protX. The first is termed GB/ACE (Schaefer & Karplus, \textit{J. Phys. Chem.}, 1996, 100:1578), for ‘Analytical Continuum Electrostatics’; the second is termed GB/HCT, for ‘Hawkins, Cramer & Truhlar’ (HCT, \textit{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 protX. The last section, Installation and Testing, describes the source file organization, the method to merge the GB source code with an existing protX distribution, and the execution of test files.
14.2 Theory

14.2.1 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 [38]. The electrostatic energy $E_{\text{elec}}$ is the sum of the Coulomb interaction energies between all solute charges and a solvation term $\Delta E_{\text{solv}}$; 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:

$$E_{\text{elec}} = \sum_{i<j} q_i q_j/r_{ij} + \Delta E_{\text{solv}}$$  \hspace{1cm} (14.1)

$$\Delta E_{\text{solv}} = \sum_i \Delta E_{\text{self}}^i + \sum_{i<j} \Delta E_{\text{int}}^{ij}.$$  \hspace{1cm} (14.2)

In the GB model, the solvent contribution $\Delta E_{\text{int}}^{ij}$ to the interaction energy between the charges $q_i$ and $q_j$ is approximated by [30]:

$$\Delta E_{\text{int}}^{ij} = -\frac{\tau q_i q_j}{(r_{ij}^2 + b_i b_j \exp[-r_{ij}^2/4b_i b_j])^{1/2}}$$  \hspace{1cm} (14.3)

where $r_{ij}$ is the distance between the charges, $\tau$ is given by

$$\tau = 1 - 1/\epsilon_w,$$  \hspace{1cm} (14.4)

$\epsilon_w$ 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

$$\Delta E_{\text{self}}^i = -\frac{\tau q_i^2}{2b_i},$$  \hspace{1cm} (14.5)

where $\Delta E_{\text{self}}^i$ is the self-energy of charge $i$. By partitioning the solute into atomic volumes (following Lee & Richards, for example [39]), one can express the self-energy $\Delta E_{\text{self}}^i$ as a sum over all the solute atoms [31, 32]:

$$\Delta E_{\text{self}}^i = -\frac{\tau q_i^2}{2R_i} + \tau q_i^2 \sum_{k \neq i} E_{\text{self}}^{ik},$$  \hspace{1cm} (14.6)

where $R_i$ is a constant atomic radius to be determined (close to the van der Waals radius) and $E_{\text{self}}^{ik}$ 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 $i$ [40].

The volume integral $E_{\text{ik}}$ is approximated in two steps. The first step is to approximate the electric field by the ‘Coulombic field’ of charge $i$ [40]. This is simply
14.2. THEORY

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 [32]:

\[
E_{ik}^{\text{self}} = \frac{1}{\omega_{ik}} \exp\left(-r_{ik}^2/\sigma_{ik}^2\right) + \frac{V_k}{8\pi} \left(\frac{r_{ik}^3}{r_{ik}^4 + \mu_{ik}^4}\right)^4.
\] (14.7)

Here, \( \omega_{ik} \) and \( \mu_{ik} \) 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 \( \alpha \) 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 \( \alpha \). Ionic strength is not included, although methods to do so have been proposed [41, 42]. Volumes \( V_k \) can be either calculated using Voronoi polyhedra (using an external program [39] and reading them into protX), or assigned values from existing libraries [32, 41, 43]. 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, \( \Delta E_i^{\text{self}} \) can sometimes become positive, so that the (necessarily positive) solvation radius can no longer be defined by Eq. (14.5). Therefore, we use a definition proposed by Schaefer et al. [44]:

\[
b_i = -\frac{\tau q_i^2}{2\Delta E_i^{\text{self}}} \quad \text{if } \Delta E_i^{\text{self}} \leq E_{\text{min}} = -\frac{\tau q_i^2}{2b_{\text{max}}} \\
= b_{\text{max}} \left(2 - \frac{\Delta E_i^{\text{self}}}{E_{\text{min}}}\right) \quad \text{if } \Delta E_i^{\text{self}} \geq E_{\text{min}}
\] (14.8)

Here, \( b_{\text{max}} \) 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.

14.2.2 Calculation of forces

Interaction energy term
We first consider the GB ‘interaction’ term, on the far right of Eq. (14.2), and its gradient \( \nabla_n \) 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:

\[
\nabla_n \sum_{i<j} \Delta E_{ij}^{\text{int}} = \sum_{i<j} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial r_{ij}} \nabla_n r_{ij} + \sum_{i<j} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_i} \nabla_n b_i + \sum_{i<j} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_j} \nabla_n b_j
\] (14.9)
Only terms with \( i = n \) or \( j = n \) contribute to the first sum on the right. The second sum can be written

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

(14.10)

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:

\[
\nabla_n \Delta E_i^{\text{self}} = \sum_{k \neq i} \nabla_n E_{ik}^{\text{self}} = \nabla_n E_{in}^{\text{self}} \quad \text{if} \quad i \neq n
\]

\[
= \sum_{k \neq n} \nabla_n E_{nk}^{\text{self}} \quad \text{if} \quad i = n.
\]

(14.11)

Grouping the second and third terms on the right of (14.9) and rearranging the first, we obtain:

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

(14.12)

with

\[
dE_i^{\text{int}, b} = \sum_{j \neq i} \frac{\partial \Delta E_{ij}^{\text{int}}}{\partial b_i}
\]

(14.13)

\[
\frac{\partial b_n}{\partial \Delta E_i^{\text{self}}} = -\frac{b_n}{\Delta E_i^{\text{self}}} \quad \text{if} \quad \Delta E_i^{\text{self}} \leq E_{min} = -\frac{\tau q_i^2}{2b_{max}}
\]

\[
= -\frac{b_{max}}{E_{min}} \quad \text{if} \quad \Delta E_i^{\text{self}} \geq E_{min}
\]

(14.14)

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 (14.12), the derivatives of \( \Delta E_{in}^{\text{int}} \) are the same for GB/ACE and GB/HCT:

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

(14.15)

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

(14.16)

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

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

(14.17)
The parameters $\omega_{ij}$, $\sigma_{ij}$, $\mu_{ij}$ are defined by:

\[
\frac{1}{\omega_{ik}} = \frac{4}{3\pi \alpha_{ik}^3} \left( Q_{ik} - \arctan Q_{ik} \right) \frac{1}{\alpha_{ik} R_k} \quad (14.18)
\]

\[
\sigma_{ik}^2 = \frac{3(Q_{ik} - \arctan Q_{ik})}{(3 + f_{ik})Q_{ik} - 4\arctan Q_{ik}} \alpha_{ik}^2 R_{ik}^2 \quad (14.19)
\]

\[
Q_{ik} = \frac{q_{ik}^2}{(2q_{ik}^2 + 1)^{1/2}} \quad (14.20)
\]

\[
f_{ik} = \frac{2}{q_{ik}^2 + 1} - \frac{1}{2q_{ik}^2 + 1} \quad (14.21)
\]

\[
q_{ik}^2 = \frac{\pi}{2} \left( \frac{\alpha_{ik} R_k}{R_i} \right)^2 \quad (14.22)
\]

\[
\alpha_{ik} = \text{Max}(\alpha, R_i/R_k) \quad (14.23)
\]

\[
\mu_{ik} = \frac{77\pi\sqrt{2} R_i}{512(1 - 2\pi^3/2\sigma_{ik}^3)} \frac{R_i}{\omega_{ik} V_k} \quad (14.24)
\]

\[
V_k = \frac{4}{3} \pi R_k^3 \quad (14.25)
\]

**GB/HCT self-energy**  
With GB/HCT, the self-energy contribution $E_{\text{self}}^\text{ik}$ is given by [31]

\[
4E_{\text{self}}^\text{ik} = \frac{1}{L_{ik}} - \frac{1}{U_{ik}} + \frac{r_{ik}}{4} \left( \frac{1}{U_{ik}^2} - \frac{1}{L_{ik}^2} \right) + \frac{1}{2r_{ik}} \ln \frac{L_{ik}}{U_{ik}} + \frac{R_k^2}{4r_{ik}} \left( \frac{1}{L_{ik}^2} - \frac{1}{U_{ik}^2} \right), \quad (14.26)
\]

where

\[
L_{ik} = 1 \quad \text{if} \quad r_{ik} + R_k \leq R_i,
\]

\[
L_{ik} = R_i \quad \text{if} \quad r_{ik} - R_k \leq R_k < r_{ik} + R_k,
\]

\[
L_{ik} = r_{ik} - R_k \quad \text{if} \quad R_i \leq R_k < r_{ik} - R_k, \quad (14.27)
\]

\[
U_{ik} = 1 \quad \text{if} \quad r_{ik} + R_k \leq R_i,
\]

\[
U_{ik} = r_{ik} - R_k \quad \text{if} \quad R_i < r_{ik} + R_k. \quad (14.28)
\]

The corresponding gradient is given by:

\[
\frac{4}{r_{ik}} \frac{\partial E_{\text{self}}^\text{ik}}{\partial r_{ik}} = -\frac{1}{r_{ik}} \left( \frac{L'_{ik}}{L_{ik}^2} - \frac{U'_{ik}}{U_{ik}^2} \right) + \frac{1}{4r_{ik}} \left( \frac{1}{U_{ik}^2} - \frac{1}{L_{ik}^2} \right) - \frac{1}{2} \left( \frac{U'_{ik}}{U_{ik}^3} - \frac{L'_{ik}}{L_{ik}^3} \right) \quad (14.29)
\]

\[
-\frac{1}{2r_{ik}^3} \ln \frac{L_{ik}}{U_{ik}} + \frac{1}{2r_{ik}^2} \left( \frac{L'_{ik}}{L_{ik}^3} - \frac{U'_{ik}}{U_{ik}^3} \right) - \frac{R_k^2}{4r_{ik}^3} \left( \frac{1}{L_{ik}^2} - \frac{1}{U_{ik}^2} \right) - \frac{R_k^2}{2r_{ik}^3} \left( \frac{L'_{ik}}{L_{ik}^3} - \frac{U'_{ik}}{U_{ik}^3} \right)
\]

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. (14.25), 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 [31].
This basic model was modified by Onufriev et al. \cite{41} to improve performance for proteins. The self-energy in Eq. (14.6) is replaced by:

$$\Delta E_{i}^{\text{self}} = -\frac{\tau q_i^2}{2b_i}$$ (14.30)

$$b_i = \left[ (R_i - \rho_0)^{-1} - \lambda \sum_{k \neq i} E_{ik}^{\text{self}} \right]^{-1} - \delta$$ (14.31)

In other words, the atomic radius $R_i$ is reduced by a constant offset $\rho_0$, the self-energy contribution $E_{ik}^{\text{self}}$ is scaled by a constant factor $\lambda$, and the solvation radius $b_i$ is reduced by a constant offset $\delta$. The values $\lambda = 1.4$, $\rho_0 = 0.09$ Å and $\delta = 0.15$ Å were used in \cite{41}.

### 14.2.3 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, protX views 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:

$$P \backslash \{S_1, S_2\} \leftrightarrow P \backslash \{S_1, S_2\}$$
$$P \backslash \{S_1, S_2\} \leftrightarrow S_1; \text{ weight of } 1/2$$
$$P \backslash \{S_1, S_2\} \leftrightarrow S_2; \text{ weight of } 1/2,$$

where $P \backslash \{S_1, S_2\}$ 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 protX through the \textbf{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:

\[
\Delta E^{\text{int}} = \frac{1}{2} \sum_{p=1}^{N} w_p \left( \sum_{i \in A_p, j \in B_p} \Delta E_{ij}^{\text{int}} \right) \tag{14.32}
\]

\[
\Delta E^{\text{self}} = \sum_{p=1}^{N} w_p \sum_{i \in A_p, j \in B_p} \left( -\frac{\tau q_i^2}{2R_i} \delta_{ij} + \tau q_i^2 E_{ij}^{\text{self}} \right) \tag{14.33}
\]

These equations generalize Eqs. (14.3), (14.6), which correspond to a single pair \(P_1 = M \times M\), \(M\) being the whole macromolecule. The factor \(\frac{1}{2}\) in Eq. (14.32) corrects for double counting of \(i, j\) and \(j, i\) terms; \(\delta_{ij}\) is the Kronecker symbol.

### 14.2.4 Crystal symmetry

The GB model has been implemented for systems with symmetry (crystallographic or otherwise); for details, see Moulinier et al, 2003 [9].

### 14.3 Syntax

#### 14.3.1 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. 14.2):

\[
E_{\text{GBSOLV}} = E_{\text{GBSELF}} + E_{\text{GBINT}}
\]

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.

#### 14.3.2 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 Excusive 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 $\alpha$ in Eq. (14.23). Default: 1.

LAMBDA=$<$real$>$ Scaling factor for solvation radii in GB/HCT; denoted $\lambda$ in Eq. (14.31). Default: 1.

OFFSET$=$<real> Offset for atomic radii in GB/HCT; denoted $\rho_0$ in Eq. (14.31). Default: 0.

14.3.3 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 paramber.gb.inp, located in $GBPROTX/gbtoppar$ (see File Organization, below). These volumes are automatically read along with the other force field parameters. The first set was developed by Schaefer and coworkers [43] and modified and tested for protein simulations by Calimet et al [45], 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 [41] and is meant to be used with the Amber all-atom force field [46]. Other volume libraries are available in the literature and can be formatted for protX, for example nucleic acid libraries [47].

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 $\epsilon$, $\sigma$; the second pair is $\epsilon$, $\sigma$ for 1–4 non-bonded interactions. The last two reals are $V$, the atomic volume (Eqs. 14.6, 14.25), and $S$, the scaling parameter used for the HCT solvation radius (see text following Eq. 14.29). 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 protX 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 [48]. 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 [43],
then reduced by a factor of 0.9 to account for systematic errors in the GB/ACE
self-energy approximation [45]. Several programs have the capability to calculate
Voronoi volumes for each individual atom of a particular protein (e.g. 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 protX 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 vector do (rmsd = wmain) (all) {copy into rmsd field}
4 flags exclude * include gbse gbin end {activate GB energy terms}

5 parameter reduce selection=(all) {average volumes over}
6 overwrite=true mode=average end {each chemical type}
7 end
8 flags include bonds angl dihe impr vdw elec end {reactivate}
```

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

### 14.3.4 Examples

**Minimization with GB/ACE**

```
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 EPS=1. WEPS=80. smooth=1.3 gbace {GB options}
8 end
9 end
10 flags include gbse gbin end
```
Molecular dynamics with GB/HCT

remarks Asparagine MD with GB/HCT
remarks this file: dyna.inp

topology
@PROTX: gbttoppar/amber/topamber.inp !Amber topology file
@PROTX: gbttoppar/amber/patches.pro !N- and C-terminal patches
end !for Amber force field

parameter
@PROTX: gbttoppar/amber/paramber.gb.inp {Amber parameter file }
end {with GB parameters}

segment
name="ASN1"
molecule name=ASN number=1 end
end
patch NASN refe=nil=(resid 1) end
patch CASN refe=nil=(resid 1) end

parameter
nbonds
atom cdie trunc
e14fac=0.8333333
cutnb 500. ctonnb 480. ctofnb 490.
tolerance=100. ! only build nonbonded list once
nbxmod 5 vswitch
wmin=1.0
end
end
parameters nbonds
EPS=1. WEPS=80. GBHCT ! GB parameters
offset=0.09 lambda=1.33 ! GB parameters
end end

coor @volumes.pdb ! PDB with volumes in wmain
vector do (RMSD = wmain) (all) ! copy into rmsd
vector do (rmsd = rmsd * 0.9) (all) ! reduce volumes by 10%
flags include gbse gbin end
parameter reduce sele=(all) overwrite=true mode=average end end

coor @asn.pdb

! Now run constant energy dynamics; random initial velocities
vector do (vx = maxwell(250)) (all)
vector do (vy = maxwell(250)) (all)
vector do (vz = maxwell(250)) (all)
dynamics verlet
   nstep=500000 timest=0.001 {ps} ! 500 ps dynamics
   iasvel=current ! current velocities
   nprint=250 iprfrq=250 ! statistics output
end

stop
Chapter 15
Fluctuating Dielectric Boundary GB

15.1 Fluctuating Dielectric Boundary method

The FDB method was proposed and tested earlier [5, 49]. With FDB, we modify the GB formulation to employ “residue” solvation radii, leading to a “Residue” GB model [49]. We define a self-energy contribution corresponding to a particular residue pair $I$, $J$ by the expression

$$E_{\text{self}}^{IJ} = \sum_{i \in I, j \in J} E_{ij}^{\text{self}}, \quad (15.1)$$

where the sum extends over atom pairs where $i$ belongs to residue $I$ and $j$ to residue $J$. The self-energy of residue $I$ can be written

$$E_{I}^{\text{self}} = \sum_{J} E_{IJ}^{\text{self}}, \quad (15.2)$$

and the total self-energy can be written

$$E^{\text{self}} = \sum_{I} E_{I}^{\text{self}}. \quad (15.3)$$

We then define the residue solvation radius $B_{I}$ by the relation

$$E_{I}^{\text{self}} \overset{\text{def}}{=} \tau \sum_{i \in I} \frac{q_{i}^{2}}{2B_{I}}, \quad (15.4)$$

$B_{I}$ is a harmonic average over the $b_{i}, i \in I$, weighted by the squared charges.

We now define the contribution $g_{IJ}$ of residues $I$ and $J$ to the total screening energy $\Delta G^{\text{solv}}$:

$$g_{IJ} = \sum_{i \in I, j \in J} \tau q_{i}q_{j} \left(r_{ij}^{2} + B_{I}B_{J} \exp[-r_{ij}^{2}/4B_{I}B_{J}] \right)^{-1/2} \quad (15.5)$$
For $I = J$, the double summation in Eq. (15.5) is actually restricted to pairs of distinct atoms, $i \neq j$. For fixed interatomic distances $r_{ij}$, $g_{IJ}(B_I B_J)$ is a slowly varying function of $B \equiv B_I B_J$. This dependency can be approximated by a low-order power expansion [49]:

$$g_{IJ}(B) \approx c_{1J}^{IJ} + c_{2J}^{IJ} B + c_{3J}^{IJ} B^2 + c_{4J}^{IJ} B^{-1/2} + c_{5J}^{IJ} B^{-3/2}$$  \hspace{1cm} (15.6)

The coefficients $c_{nJ}^{IJ}$ can be pre-computed and stored in the energy matrix. To keep the notations simple, their dependency on the particular rotamer combination $r_I, r_J$ is not indicated explicitly. The quantities $B_I$ and $B_J$ can be obtained from residue pair contributions stored in the energy matrix. Thus, with (15.6), the Fluctuating Dielectric Boundary method only involves quantities that depend on residue pairs.

### 15.2 FDB implementation

For each FDB residue pair and all possible rotamer combinations, the GB interaction energy is fitted to the power expansion (15.6) in the range $B = 1–150$ Å² using protX. The code is based on the general linear fit subroutine LFIT from Numerical Recipes [49, 50]. The fit is controlled at the level of the protX script language by a statement of the form:

```
pick gbfit <selection1> <selection2>
```

which computes the GB interaction energy between two groups of atoms $R_1, R_2$ defined by the two selections, which occupy a specific conformation. The individual solvation radii of $R_1$ and $R_2$ are not computed from the protein structure but are defined by the relation $B_{R1} B_{R2} = B$. protX performs the fit and stores the fitting coefficients in the script variables `$coef1, \ldots, coef5`, which can be printed out by a script command, e.g., `display coef1`. In the energy matrix, this information is stored along with the other interaction energy terms. The contribution of each residue pair to the GB self energy is also stored as a separate item in the energy matrix. The matrix entry for a pair $I, J$ and a particular rotamer combination $r_I, r_J$ is shown in Fig. 15.1.

With the NEA method, at each step $t$ of the Monte Carlo simulation, if a residue $I$ is displaced, the resulting energy change $\Delta E(t)$ is computed from energy matrix elements of the form $E_{IJ}$. With FDB, the solvation radii change over time and so additional operations are needed:

1. Throughout the trajectory, we maintain an up-to-date list of all the residue solvation radii $B_I$, whose values fluctuate over the trajectory. This is possible since the GB self energy information is available in the matrix. At each MC
step, $B_I$ is only updated if a residue close to residue $I$ (based on a neighbor list built ahead of time) is displaced or mutated.

2. At each MC trial move, if a solvation radius $B_I$ changes, then residue $I$ will contribute to the energy change $\Delta E(t)$, since its GB interaction energies $g_{IJ}$ with all other residues $J$ are affected. In fact, the contributions to $\Delta E(t)$ that result from the change in $B_I$ are only computed for residues $J$ within a certain cutoff distance of $I$. These $J$ values are read out of a second neighbor list, built ahead of time, based on the size of the fitting coefficients $c_{n}^{IJ}$ (Eq. [15.6]): small coefficients indicate distant neighbors. For each neighbor $J$, the appropriate (rotamer-dependent) $g_{IJ}$ value is obtained from the product $B_IB_J$ and the fitting coefficients $c_{n}^{IJ}, n = 1, ..., 5$, via Eq. [15.6].

3. At regular intervals (about every 1000 MC steps), the entire energy and all the solvation radii are recomputed from scratch, to avoid propagation of numerical errors.
Diagonal matrix elements

Residue number $I$ Residue name Rotamer number $R$

<table>
<thead>
<tr>
<th>Residue number $I$</th>
<th>Residue name</th>
<th>Rotamer number $R$</th>
<th>$E_{\text{ref}}$</th>
<th>$r_{\text{vdW}}$</th>
<th>$r_{\text{elec}}$</th>
<th>ASA</th>
<th>Q2</th>
<th>$E_{\text{GB self}}$</th>
<th>$E_{\text{GB NEA}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1141</td>
<td>ARG</td>
<td>38</td>
<td>-45.24</td>
<td>24.56</td>
<td>-46.33</td>
<td>0.26</td>
<td>3.35</td>
<td>-72.60</td>
<td>39.89</td>
</tr>
</tbody>
</table>

Off-diagonal matrix elements

Residue numbers $I$, $J$

<table>
<thead>
<tr>
<th>Residue number $I$</th>
<th>Residue number $J$</th>
<th>$E_{\text{ref}}$</th>
<th>$r_{\text{vdW}}$</th>
<th>$r_{\text{elec}}$</th>
<th>ASA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2035</td>
<td>TYD 1122 HIP</td>
<td>-2.1E-02</td>
<td>2.7E-02</td>
<td>0.30E-02</td>
<td>8.5E-03</td>
</tr>
<tr>
<td>2035</td>
<td>TYD 2028 LEU</td>
<td>-0.177</td>
<td>0.16</td>
<td>4.4E-02</td>
<td></td>
</tr>
</tbody>
</table>

Figure 15.1: Energy matrix elements. For **diagonal elements** (above), we show an example of an FDB element and an NEA element (one rotamer each). The individual energy components are labelled. ASA labels the surface energy term. Although the Arg1141 position is treated with FDB, the matrix includes the NEA estimate of the GB contribution. The quantity labeled Q2 is the total squared charge $\sum_{i \in I} q_i^2$, needed to compute the solvation radius $B_I$ (Eq. 15.4). The five rightmost quantities are the fitting coefficients $c_{II}^n$ (Eq. 15.6). For **off-diagonal elements** (below), we also show an FDB and an NEA pair (one rotamer combination each). For the FDB pair, the GB self-energy contributions $E_{I,J}^{\text{self}}$ and $E_{J,I}^{\text{self}}$ (Eq. 15.1) are both stored (the GB self energy is non-symmetric).
Part V

The protX program
Chapter 16

protX language

ProtX uses the same command parser as the Xplor program, so that the Xplor manual by Axel T. Brünger can serve as a protX manual. Here, we adapt (with permission from the author) a subset of the Xplor manual, describing the essential protX features, to make the present Proteus manual self-contained.

16.1 Input format

ProtX uses a free field format. Characters between braces {} or after an “!” on the same line are ignored. The carriage return is treated as a space. Letters are converted to uppercase upon parsing. Spacing does not affect the parsing.

16.2 Symbols

A symbol is a word with a “$” as the first character. It is replaced by another word that has been assigned by the user or protX during execution. Symbols can be assigned and manipulated by the evaluate statement and several control statements (see Section 16.3). The data can be real or string. ProtX declares certain symbols internally when certain statements are executed. For example:

$? produces a list of the currently assigned symbols.

$ANGL, $BOND, $DIHE, $ELEC, $ENER, $HARM, $IMPR, $PLAN, $VDW represent partial energy terms (see Sections 18.4 and 18.5). They are declared upon evaluation of the energy function.

$GRAD is the norm value of the energy gradient. It is declared upon energy calculation.
$RESULT$ contains the result after execution of certain statements, such as “VEC-Tor SHOW” and “COORdinates RMS END”.

$TIME$ contains the wall-clock time (string).

Below, a symbol $NEWSYMBOL$ is declared, used, then redeclared as a character string.

\[
\text{evaluate (}$NEWSYMBOL=3.40+433^2)\]
\[
\text{evaluate (}$GARBAGE=\sqrt{NEWSYMBOL})\]
\[
\text{evaluate (}$NEWSYMBOL=\text{“testing 1 2 3”})\]

A special case of a word is a wildcard.

< wildcard > ::= \{ \ast | \% | \# | + | < string > \}

  * matches any string.
  % matches a single character.
  # matches any number.
  + matches any digit.

16.3 Control statements

In protX, we can distinguish control statements and application statements. Control statements allow loops, conditional tests, switching the input stream to another file, opening and closing files, and various other operations:

< control-statement > ::= 

  @< filename > Switches the input stream to a file (the initial stream is standard input). At the end of the file, the parsing stream is switched back to the previous input.

  @@< filename > has the same effect, except when the stream file is invoked within a structured loop statement. In this case, the “@” statement inserts the contents of file filename into the loop and removes the statement in subsequent loop cycles, whereas the “@@” statement reads from filename each time the loop hits the statement.

  DISPLAY < record > writes the record to a file specified by the “SET DISPLAY” statement. The record can be any sequence of characters terminated by a carriage return.

  EVALuate < evaluate-statement > manipulates symbols (see Section 16.7).
FOR <symbol> IN ( { <word> } ) <basic-loop> executes the statements within the basic loop.

FOR <symbol> IN ID <selection> <basic-loop> assigns the symbol to the internal atom identifier for all atoms in the selection, and executes the statements within the basic loop.

IF <condition> THEN <protX-statement>
   [ { ELSEIF <condition> THEN <protX-statement> }]
   [ ELSE <protX-statement> ] END IF is the IF statement.

REMARKS <record> writes the record to an internal title store. The record can be any sequence of characters terminated by a carriage return. It can contain symbols that are substituted before the record is stored. The internal title store is written to the first lines of output files.

SET <set-statement> END sets various global parameters and options (see Section 16.6).

WHILE <condition> <basic-loop> executes a while loop.

<condition> ::= ( <word> = |#| > | < |GE|LE <word> ) compares two symbols.

<basic-loop> ::= LOOP <label> { <protX-statement>
   [EXIT <label>] } END LOOP <label> represents a loop. The label is a string with up to four characters. Loops may be nested.

This example writes the characters a, b, c, d, e to a file:

set display=test.dat end
for $1 in ( a b c d e ) loop main
   display $1
end loop main

When protX executes loops, it stores all input information in internal buffers. To avoid this, one should use the “@@” statement:

coordinate disposition=comp @reference.pdb
for $fil in ( "coor1.pdb" "coor2.pdb" "coor3.pdb" "coor4.pdb" ) loop main
   coordinate @@$fil
   coordinate rms end
end loop main
16.4 Abbreviations

In most cases, keywords and qualifiers can be abbreviated to four characters.

16.5 Input and output

Most I/O files are ASCII files. The record length is not more than 132 characters.

16.6 Set statement

The set statement allows one to change certain parameters that control execution and output:

\[ <\text{set-statement}> := \]

- \( \text{DISPlay-file}= <\text{filename}> \) specifies an output file for DISPlay statements (Section 16.3).
- \( \text{ECHO}=\text{ON|OFF} \) determines whether the input stream will be echoed to standard output.
- \( \text{MESSage}=\text{OFF|NORMal|ALL} \) determines messages verbosity.
- \( \text{PRINt-file}= <\text{filename}> \) specifies an output file for PRINt statements (default: OUTPUT).
- \( \text{SEED}= <\text{real}> \) is a seed for the internal random-number generator. It can be any positive number.

16.7 Evaluate statement

The evaluate statement allows one to manipulate symbols and constants.

\( \text{EVALuate} (<\text{evaluate-statement}> ) \) is a control statement.

\[ <\text{evaluate-statement}> := <\text{symbol}> = <\text{operation}> \]

\[ <\text{operation}> := <\text{vflc}> [<\text{op}> <\text{operation}> ] \]

\[ <\text{op}> := \]

+ denotes addition or concatenation for strings.
- denotes subtraction.
* denotes multiplication.
/ denotes division.
^ denotes exponentiation.
** denotes exponentiation.

\(<vflc>\):= \(<\text{function}>\) | \(<\text{symbol}>\) | \(<\text{real}>\) | \(<\text{integer}>\) | \(<\text{string}>\)  The available functions, such as SIN, COS are defined in Section 16.9

Some examples:

EVALuate ($x=1.0$)

EVALuate ($x=x+2.2$)

EVALuate ($x=x*COS(2.*3.14)$)

### 16.8 Atom Selection

protX has a powerful atom selection syntax. The number of selected atoms from the last executed selection statement is stored in the symbol $SELECT$. Information associated with atom selections is lost when changing the molecular structure. However, certain selections are protected by mapping the selected atoms to the new molecular structure after it has been modified. This applies to all atom properties (Section 16.9) except for the internal stores, which are fragile. It also applies to the atom-based parameter statements (Section 17.2.1).

\(<\text{selection}>\):= ( \(<\text{selection-expression}>\) )

\(<\text{selection-expression}>\):= 

\(<\text{term}>\) selects atoms that belong to the term.

\(<\text{term}>\) \{ OR \(<\text{term}>\) \} selects all atoms that belong to either one of the terms.

\(<\text{term}>\):= 

\(<\text{factor}>\) selects atoms that belong to the factor.

\(<\text{factor}>\) \{ AND \(<\text{factor}>\) \} selects all atoms that belong to all of the factors.

\(<\text{factor}>\):= 

( \(<\text{selection-expression}>\) ) selects all atoms that are selected in selection expression.
ALL selects all atoms.

\(<factor>\) AROUND \(<real>\) selects all atoms that are within the specified real cutoff value around any selected atom in the factor.

ATOM \(<*segment-name*>\) \(<*residue-number*>\) \(<*atom*>\) selects all atoms that match the specified segment name, residue number, and atom name or wildcards of them.

ATTRIBUTE [ABS] \(<property>\) \(<|=|\#|>\) \(<real>\) selects all atoms that have (absolute) properties less than, equal to, not equal to, or greater than the specified real number.

BYGROUP \(<factor>\) selects all atoms that belong to groups (see Section 17.1.1) containing at least one atom that has been selected in the factor.

BYRes \(<factor>\) selects all atoms that belong to residues containing at least one atom that has been selected in the factor.

CHEMical \(<*type*>\) selects all atoms that match the specified type (Section 17.1.1) or a wildcard of it.

CHEMical \(<type>:<type>\) selects all atoms that have types greater than or equal to the first type but less than or equal to the second type in alphanumeric order.

ID \(<integer>\) selects all atoms that match the specified internal atom number. It should be used with caution. The main application is in conjunction with the “FOR \(<symbol>\) IN ID” statement (Section 16.3).

KNOWn selects all atoms with known coordinates.

NAME \(<*atom*>\) selects all atoms that match the specified atom name (Section 17.1.1) or a wildcard of it.

NAME \(<atom>:<atom>\) selects all atoms that have atom names greater than or equal to the first atom name but less than or equal to the second atom name.

NOT \(<factor>\) selects all atoms that have not been selected in the factor.

POINT \(<3d-vector>\) CUT \(<real>\) selects all atoms that are within the specified real cutoff value around the specified 3d-vector.

PREVIOUS selects all atoms that have been selected in a previous selection in application statements that contain multiple selections.

RESIdue \(<*residue-number*>\) selects all atoms that match the specified residue number (Section 17.4) or a wildcard of it.
RESIDue <residue-number>:<residue-number> selects all atoms that have residue numbers greater than or equal to the first residue number but less than or equal to the second residue number.

RESName <*residue-name*> selects all atoms that match the specified residue name (Section 17.1.1) or a wildcard of it.

RESName <residue-name>:<residue-name> selects all atoms that have residue names greater than or equal to the first residue name but less than or equal to the second residue name.

<factor> SAROund <real> selects all atoms that are within the specified real cutoff value around any selected atom in the factor.

SEGIdentifier <*segment-name*> selects all atoms that match the specified segment name (Section 17.4) or a wildcard of it.

SEGIdentifier <segment-name>:<segment-name> selects all atoms that have segment names greater than or equal to the first segment name but less than or equal to the second segment name.

STORE1 | STORE2 | STORE3 | STORE4 | STORE5 | STORE6 | STORE7 | STORE8 | STORE9 selects all atoms for which the value of STOREi is greater than 0; e.g., STORE2 is short hand for “ATTRibute STORE2 > 0”, etc. The STOREi can be defined by the vector ID statement or the vector statement (see Section 16.9).

TAG selects exactly one atom from each residue. These selected atoms may be used to “tag” all residues without having to refer to residue numbers or identifiers. The sequence of selected atoms is determined by the order in which the residues have been created through the segment statement (see Section 17.4).

[property]:== B | BCOMp | CHARge | DX | DY | DZ | FBETa | HARM | MASS | Q | QCOMp | REFX | REFY | REFZ | RMSD | VX | VY | VZ | X | XCOMp | Y | YCOMp | Z | ZCOMp is a group of properties defined in Section 16.9.

For example:

( name ca and resid 40:100 ) ! residues 40 to 100
( resname phe and ( residue 1 around 20.0 ) ) ! Phe residues near residue 1
( byresidue ( resname phe and ( residue 1 around 20.0 ) ) ) ! selects entire residue
16.9 Vector statement

The vector statement allows one to manipulate atomic properties, such as masses, charges, coordinates, forces, and atom names. Mathematical expressions can be constructed that involve atomic properties. The vector statement can also be used to define and store an atom selection for later use.

16.9.1 Syntax

VECTor <$vector-statement$> is invoked from the main level of protX.

<$vector-statement$>::=

 $vector-mode$ <$vector-expression$> <$selection$>

$vector-mode$>::=

 DO manipulates atom properties for all selected atoms.
 IDENtify defines and stores an atom selection.
 SHOW <$vector-show-property$> can be used to display atom properties.

$vector-show-property$>::=

 AVE shows the arithmetic mean and stores it in $RESULT$.
 ELEMent shows selected elements and stores the last one in $RESULT$.
 MAX shows the maximum of selected elements and stores it in $RESULT$.
 MIN shows the minimum of selected elements and stores it in $RESULT$.
 NORM shows the norm $\sqrt{\sum x_i^2}$ and stores it in $RESULT$.
 RMS shows the rms deviation and stores it in $RESULT$.
 SUM shows the sum of selected elements and stores it in $RESULT$.

$vector-expression$>::=

 $atom-property$ = $vector-operation$ carries out the vector operation and assigns it to the specified atom property.
 $vector-operation$ carries out the vector operation without assigning the result to an atom property. It should be used for the IDENtify and SHOW vector modes.

$vector-operation$>::=

Operators with the same precedence are executed from left to right. The data type of the operands has to match the operation. The following is a list of the operators with increasing precedence:

\(<\text{op}>:==\)

- addition; concatenation for strings
- subtraction
- multiplication
- division
- exponentiation
- exponentiation

\(<\text{vflc}>:==\)

- \(<\text{atom-property}>\mid<\text{function}>\mid<\text{integer}>\mid<\text{real}>\mid<\text{string}>\mid<\text{symbol}>\) is a group of functions. Use of a string requires enclosure in double quotes " ". The data type of the function arguments has to match the data type of the operands.

\(<\text{function}>:==\)

\(\text{ABS}(<\text{vflc}>)\) expects one argument and returns its absolute value. Argument restrictions: no string.

\(\text{ACOS}(<\text{vflc}>)\) denotes arc cosine. Argument restrictions: no string or complex; expects argument in degrees.

\(\text{ASIN}(<\text{vflc}>)\) denotes arc sine. Argument restrictions: no string or complex; expects argument in degrees.

\(\text{COS}(<\text{vflc}>)\) denotes cosine. Argument restrictions: no string; expects argument in degrees.

\(\text{DECODE}(<\text{vflc}>)\) converts a character string to a numerical number if possible.

\(\text{ENCODE}(<\text{vflc}>)\) converts a numerical number to a character string.

\(\text{EXP}(<\text{vflc}>)\) is an exponentiation function. Argument restrictions: no string.

\(\text{GAUSS}(<\text{vflc}>)\) is a Gaussian distribution random-number function; it has one argument, the desired standard deviation. The mean of the distribution is always zero. Argument restrictions: no string or complex.
HEAVY(<vflc>) is a heavy-side function; it expects one real-number argument. If the argument is greater than zero, heavy returns a one; otherwise heavy returns a zero. Argument restrictions: no string or complex.

INT(<vflc>) is a truncation. Argument restrictions: no string.

LOG10(<vflc>) is a natural log. Argument restrictions: no string or complex; argument must be greater than zero.

LOG(<vflc>) is a logarithmic function. Argument restrictions: no string or complex; real numbers must be greater than zero, and the complex (0.0,0.0) is illegal.

MAX(<vflc> {, <vflc> }) is a maximum-value function; it must have at least two arguments, and it returns the value of the argument with the maximum value. Argument restrictions: no string or complex.

MAXW(<vflc>) is a Maxwellian distribution random-number function; it has one argument, the desired standard deviation. The mean of the distribution is always zero. Argument restrictions: no string or complex.

MIN(<vflc> {, <vflc> }) is a minimum-value function; it must have at least two arguments, and it returns the value of the argument with the minimum value. Argument restrictions: no string or complex.

MOD(<vflc>,<vflc>) returns the remainder of the first argument divided by the second. Argument restrictions: no string or complex.

NORM(<vflc>) is a normalization function; it has one argument, which must be a recognized variable. This function calculates the sum of the squares of all selected elements in the argument array. Then it divides each selected element by the square root of the sum of the squares. Argument restrictions: no string or complex.

RANDom() is a random-number function; it has no argument. It returns a uniform distribution between 0 and 1.

SIGN(<vflc>) is a transfer of sign. If the argument is >= 0, it returns +1; if the argument is < 0, it returns −1. Argument restrictions: no string or complex.

SIN(<vflc>) denotes sine. Argument restrictions: no string; expects degrees.

SQRT(<vflc>) returns the square root of the given argument. Argument restrictions: no string; no negative real numbers.

TAN(<vflc>) denotes tangent in degrees. Argument restrictions: no string or complex.
<atom-property>:==

B  B-factors of main coordinate set in Å² (real)

BCOMp  B-factors of comparison coordinate set in Å² (real)

CHARge  electric charge in electronic charges (real)

CHEMical  chemical atom type (string)

DX  x component of first derivatives in kcal mole⁻¹ Å⁻¹ (real)

DY  y component of first derivatives in kcal mole⁻¹ Å⁻¹ (real)

DZ  z component of first derivatives in kcal mole⁻¹ Å⁻¹ (real)

FBETa  friction coefficient in psec⁻¹ (real)

HARMonic  energy constants of harmonic restraints in kcal mole⁻¹ Å⁻² (real)

MASS  mass in amu (real)

NAME  atom name (string)

Q  occupancies of main coordinate set (real)

QCOMp  occupancies of comparison coordinate set (real)

REFX  x component of reference coordinate set in Å (real)

REFY  y component of reference coordinate set in Å (real)

REFZ  z component of reference coordinate set in Å (real)

RESId  residue number (string)

RESName  residue name (string)

RMSD  array used by various modules, e.g., the COOR RMS statement

SEGId  segment or chain identifier (string)

STORE1  1st internal store, is fragile (real)

STORE2  2nd internal store, is fragile (real)

STORE3  3rd internal store, is fragile (real)

STORE4  4th internal store, is fragile (real)

STORE5  5th internal store, is fragile (real)

STORE6  6th internal store, is fragile (real)

STORE7  7th internal store, is fragile (real)

STORE8  8th internal store, is fragile (real)

STORE9  9th internal store, is fragile (real)
CHAPTER 16. PROTX LANGUAGE

\[ \text{VX} \] x component of current velocities in Å psec\(^{-1}\) (real)
\[ \text{VY} \] y component of current velocities in Å psec\(^{-1}\) (real)
\[ \text{VZ} \] z component of current velocities in Å psec\(^{-1}\) (real)
\[ \text{X} \] x component of main coordinate set in Å (real)
\[ \text{XCOMp} \] x component of comparison coordinate set in Å (real)
\[ \text{Y} \] y component of main coordinate set in Å (real)
\[ \text{YCOMp} \] y component of comparison coordinate set in Å (real)
\[ \text{Z} \] z component of main coordinate set in Å (real)
\[ \text{ZCOMp} \] z component of comparison coordinate set in Å (real)

16.9.2 Examples

The first example divides the coordinate array \(Z\) by the derivative array \(DX\), adds the quotient to the coordinate array \(Y\), and stores the result in the coordinate array \(X\). The operations are carried out component by component for all atoms.

\[ \text{vector do ( X = Y + Z / DX ) ( all )} \]

The next example computes a Gaussian distribution with standard deviation 1.0 and stores the result in the coordinate array \(x\) for all \(C^\alpha\) atoms:

\[ \text{vector do ( X = GAUSS( 1.0 ) ) ( name ca )} \]

The next example provides a listing of the \(X\) coordinates of all Tyr residues:

\[ \text{vector show element ( X ) ( resname tyr )} \]

The next example computes the average of all electric charges in residue 34. This average value is then stored in the symbol \$1\) by using the evaluate statement.

\[ \text{vector show ave ( charge ) ( residue 34 )} \]
\[ \text{evaluate ( $1 = $RESULT) } \]

The next example stores the specified atom selection in the array STORE1:

\[ \text{vector identity ( store1 ) ( attribute mass > 30.0 )} \]

The array STORE1 can be recalled by using

\[ ( \text{store1} ) \]

in a selection statement.
Chapter 17

Topology, Parameters, Structure

17.1 Topology Statement

The topology consists of a library of fragments such as amino acids, with information about atoms and their connectivity. Each atom has a name, a chemical type, and a charge. The topology is used by the segment statement to generate the 2D structure of the system. Parameters for the energy function (below) can be assigned based on chemical types or assigned to individual atoms (parameter statement, Section 17.2.1).

17.1.1 Syntax

TOPOlogy {<topology-statement>} END is invoked from the main protX level.

<topology-statement>:=

AUTOgenerate ANGLE=<logical> DIHEdral=<logical> END automatically generates all possible bond angles based on the connectivity list of the particular residue.

RESIdue <residue-name> { <residue-statement> } END adds a residue to the topology database.

PRESidue <residue-name> { [ ADD | DELEte | MODIfy ] <residue-statement> } END adds a patch residue to the topology database.

<residue-statement>:=

ANGLE <atom> <atom> <atom> adds a bond angle made by the three atoms. It should not be used if autogenerate angles are active.
ATOM [<patch-character>] <atom> <atom-statement> END adds an atom, defined by a 4-character name, a type, and charge. The patch character is a 1-character string and may be used only for PRESideue.

BOND <atom> <atom> adds a covalent bond between the specified atoms.

DIHENedral <atom> <atom> <atom> [MULTiple <integer>] adds a dihedral angle. The statement should not be used if autogenerate dihedrals are active, except for multiple dihedral. MULTiple specifies \( m \) dihedral angle entries for the same set of four atoms (Eq. 18.4). It must be accompanied by a corresponding DIHENedral angle parameter entry with appropriate multiplicity; see Eq. 18.4.

GROUP partitions the atoms into groups.

IMPRoper <atom> <atom> <atom> <atom> [MULTiple <integer>] adds an improper angle; see Eq. 18.5.

<atom>= is the name of the atom

<atom-statement>=

CHARge=<real> specifies a charge.

EXCLude=( { <atom> } ) specifies explicit nonbonded interaction exclusions.

TYPE=<type> specifies the chemical atom type, a string with up to four characters.

<type>= is any sequence of four characters.

17.1.2 Example: topology of a leucine

TOPOlogy
RESIdue LEU
GROUP
ATOM N TYPE=NH1 CHARge=-0.35 END
ATOM H TYPE=H CHARge= 0.25 END
ATOM CA TYPE=CH1E CHARge= 0.10 END
ATOM CB TYPE=CH2E CHARge= 0.00 END
ATOM CG TYPE=CH1E CHARge= 0.00 END
ATOM CD1 TYPE=CH3E CHARge= 0.00 END
ATOM CD2 TYPE=CH3E CHARge= 0.00 END
ATOM C TYPE=C CHARge= 0.55 END
17.2 Parameter Statement

The parameter statement specifies parameters for the energy function (Section 18.1). Lengths are in Å, energies in kcal mole$^{-1}$, and charges in units of the proton charge. ProtX stores and manipulates “type-based” and “atom-based” parameters. A type-based parameter is characterized by the chemical types of the atoms involved; an atom-based parameter is characterized by the individual atoms involved. The chemical types are specified in the topology statement (Section 17.1.1). Atom-based parameters always take precedence over type-based ones. Atom-based parameters can be changed or added at any time. Type-based parameters cannot be manipulated, but additional entries can be added or the database erased and reinitialized. The parameter specifications are insensitive to the atom order:

$$\text{bond a b 10.0 1.0}$$

and

$$\text{bond b a 10.0 1.0}$$

are equivalent.

17.2.1 Syntax

\text{PARAMeter } \{<\text{parameter-statement}>\} \text{ END} \text{ to invoke from the main level}
\[ \text{ANGLe} \quad \text{<type>} \quad \text{<type>} \quad \text{<type>} \quad \text{<real>} \quad \text{<real>} \quad \text{[UB} \quad \text{<real>} \quad \text{<real}>] \]

adds a bond angle parameter set for the three atom types to the parameter database. The first real specifies \( k_\phi \), in kcal mole\(^{-1}\) rad\(^{-2}\), and the second real specifies \( \theta_0 \), the equilibrium angle, in degrees. The optional UB specification activates the Urey-Bradley term (Eq. 18.3), where the first real is the Urey-Bradley energy constant \( k_{ub} \) and the second is the Urey-Bradley equilibrium distance \( r_{ub} \) between the first and the third atom that define the angle. If UB is not specified, the Urey-Bradley equilibrium distance and energy constant default to zero.

\[ \text{ANGLe} \quad \text{<selection>} \quad \text{<selection>} \quad \text{<selection>} \quad \text{<real>} \quad \text{<real>} \quad \text{[UB} \quad \text{<real>} \quad \text{<real}>] \]

is an atom-based version of the ANGLe statement.

\[ \text{BOND} \quad \text{<type>} \quad \text{<type>} \quad \text{<real>} \quad \text{<real>} \]

adds a covalent bond parameter set for the two atom types to the parameter database. The first real specifies \( k_b \) in units of kcal mole\(^{-1}\) Å\(^{-2}\), and the second real specifies \( r_0 \), the equilibrium bond length in Å.

\[ \text{BOND} \quad \text{<selection>} \quad \text{<selection>} \quad \text{<real>} \quad \text{<real>} \]

is an atom-based version of the BOND statement.

\[ \text{DIHEdral} \quad \text{<type>} \quad \text{<type>} \quad \text{<type>} \quad \text{<type>} \quad \text{[MULT} \quad \text{<integer>} \quad \text{]} \quad \{ \text{<real>} \quad \text{<integer>} \quad \text{<real>} \quad \}\]

adds a dihedral angle parameter set for the four atom types to the parameter database (see also Eq. 18.4). The MULT option specifies the multiplicity \( m \) of the dihedral angle (default: \( m=1 \)). For multiple dihedrals of multiplicity \( m \), there are \( m \) groups of 3 items following the MULT <integer> statement. The first real of each group specifies \( k_\theta \), the integer is the periodicity \( n \), and the second real specifies \( \delta \), the phase-shift angle, in degrees. If the periodicity \( n \) is greater than 0, \( k \) has the units of kcal mole\(^{-1}\); if the periodicity is 0, \( k \) has the units of kcal mol\(^{-1}\) rad\(^{-2}\) (Eq. 18.4). The special character X acts as a wildcard. Wildcards are not allowed for multiple dihedral angles. The program automatically performs the interchange \((a \ b \ c \ d) \rightarrow (d \ c \ b \ a)\) where this is required.

\[ \text{DIHEdral} \quad \text{<selection>} \quad \text{<selection>} \quad \text{<selection>} \quad \text{<selection>} \quad \text{[MULT} \quad \text{<integer>} \quad \text{]} \quad \{ \text{<real>} \quad \text{<integer>} \quad \text{<real>} \quad \}\]

is an atom-based version of the DIHEdral statement.

\[ \text{IMPRProper} \quad \text{<type>} \quad \text{<type>} \quad \text{<type>} \quad \text{<type>} \quad \text{[MULT} \quad \text{<integer>} \quad \text{]} \quad \{ \text{<real>} \quad \text{<integer>} \quad \text{<real>} \quad \}\]

adds an improper angle parameter set for the four atom types to the parameter database.
\textbf{IMPRoper} \texttt{<selection> <selection> <selection> <selection>}

\[ \text{[MULT <integer>] \{ <real> <integer> <real> \}} \] is an atom-based version of the IMPRoper statement.

\textbf{NBFIx} \texttt{<type> <type> <real> <real> <real> <real>} adds a Lennard-Jones parameter set for the specified pair of atom types to the parameter database. The first two real numbers are the A, B coefficients (Eq. 18.6) for all nonbonded interactions except the special 1–4 interactions; the second pair of reals is for the 1–4 (NBXMod=±5) nonbonded interactions.

\textbf{NBFIx} \texttt{<selection> <selection> <real> <real> <real> <real>} is an atom-based version of the NBFIx statement.

\textbf{NONB} \texttt{<type> <real> <real> <real> <real>} adds a Lennard-Jones parameter set for pairs of atoms of the same specified type to the parameter database. The first pair of reals is $\epsilon, \sigma$ (Eq. 18.6) for all nonbonded interactions except the special 1–4 interactions; the second pair is $\epsilon, \sigma$ for the 1–4 nonbonded interactions (NBXMod=±5).

\textbf{NONB} \texttt{<selection> <real> <real> <real> <real>} is an atom-based version of the NONB statement.

\textbf{VERBose} produces a verbose listing of all atom-based parameters.

\textbf{NBONds} \{ \texttt{<nbonds-statement> } \} END applies to both electrostatic and van der Waals energy calculations. It sets up global parameters for the nonbonded interaction list generation and determines the form of subsequent nonbonded energy calculations (see Eq. 18.6).

\texttt{<nbonds-statement>}:==

\textbf{CDIE|RDIE} specifies exclusive flags: constant dielectric (Coulomb’s law) or 1/$r$-dependent dielectric (Eq. 18.10). CDIE may be used in combination with VSWItch, SHIFt, and TRUNCation (default: CDIE).

\textbf{CTOFNB=}<real> specifies the distance $r_{off}$ at which the switching function or shifting function forces the nonbonded energy to zero (Eqs. 18.6, 18.10) (default: 7.5 Å).

\textbf{CTONNB=}<real> specifies the distance $r_{on}$ at which the switching function becomes effective (Eq. 18.6) (default: 6.5 Å).

\textbf{CUTNb=}<real> specifies the nonbonded interaction cutoff $r_{cut}$ for the nonbonded list generation (default: 8.5 Å).

\textbf{E14Fac=}<real> specifies the factor $e_{14}$ for the special 1–4 electrostatic interactions (Eq. 18.11) (default: 1.0).
**EPS**=<real> specifies the dielectric constant $\varepsilon$ (Eq. 18.10) (default: 1.0).

**GROUp | ATOM** specifies exclusive flags: group by group or atom by atom cutoff for nonbonded list generation (default: ATOM).

**NBXMod**=+1|−1|+2|−2|+3|−3|+4|−4|+5|−5 Exclusion list options:

+−1 no nonbonded exclusions, that is, all nonbonded interactions are computed regardless of covalent bonds.

+−2 excludes nonbonded interactions between bonded atoms.

+−3 excludes nonbonded interactions between bonded atoms and atoms that are bonded to a common third atom.

+−4 excludes nonbonded interactions between bonded atoms, atoms that are bonded to a common third atom, and or atoms that are connected to each other through three bonds.

+−5 same as (+−3), but the 1–4 nonbonded interactions are computed using the 1–4 Lennard-Jones parameters and the electrostatic scale factor $e_{14}$ (Eqs. 18.11 and 18.12).

A positive mode value causes explicit nonbonded exclusions (see exclusion statement, Section 17.1.1) to be taken into account; a negative value causes them to be discarded (default: 5).

**SWITch|SHIFT** specifies exclusive flags: electrostatic switching or shifting. SWITch may only be used in combination with RDIE, VSWITch, and REPEl=0. SHIFt may only be used in combination with CDIE, VSWITch, and REPEl=0 (default: SHIFt).

**TOLErance**=<real> specifies the distance that any atom is allowed to move before the nonbonded list gets updated. Note: if switching or shifting functions are used, the program expects $\text{CUTNB} \geq \text{CTOFNB} + 2\text{TOLErance}$. In this way the nonbonded energy is independent of the update frequency. For the REPEl option, $\text{CUTNB} \geq r_{\text{max}} + 2\text{TOLErance}$, where $r_{\text{max}}$ is the maximum van der Waals radius. TOLErance has no influence on the TRUNcation option. (default: 0.5 Å).

**TRUNcation** turns off switching or shifting; i.e., the nonbonded energy functions are “truncated” at $\text{CUTNB}$ regardless of the values of $\text{CTONNB}$ and $\text{CTOFNB}$. All nonbonded energy terms that are included in the current nonbonded list are computed. May only be used in combination with CDIE. Note: in general, the nonbonded energy will not be conserved be-
fore and after nonbonded list updates when using TRUNcation. (default: inactive).

**VSWItch** turns on van der Waals switching. May only be used in combination with RDIE, SWITch, and REPEl=0 or in combination with CDIE, SHIFt, and REPEl=0 (default: active).

**WMIN=**<real> specifies the threshold distance for close contact warnings, i.e., a warning is issued when a pair of atoms gets closer than this distance unless the nonbonded interaction is excluded by the NBXMod option (default: 1.5 Å).

### 17.3 Topology and parameter files

This section describes the most important parameter and topology files. A pair of parameter and topology files represents a force field.

#### 17.3.1 Amber ff99SB and ff14SB

The main force field used by Proteus for CPD.

#### 17.3.2 CHARMM “top_all22*” and “par_all22*” force field

Contains parameters for proteins and nucleic acids.

#### 17.3.3 AMBER/OPLS “tophopls.pro”, “parhopls.pro” files


#### 17.3.4 Files “toph19.sol” and “param19.sol” for TIP3P water

These describe the TIP3p water model (Jorgensen et al. 1983).

### 17.4 Generating the molecular structure

The segment statement generates the molecular structure by interpreting the coordinate file to obtain the residue sequence or by explicitly specifying the residue sequence. A segment can be a polypeptide chain or a collection of residues or molecules.
17.4.1 Syntax

\(<\text{residue-number}\)\> specifies the number of a residue, a four-character string (sic).

\(<\text{segment-name}\)\> specifies the name of a segment, a four-character string.

SEGMENT { \(<\text{segment-statement}\) \} END to invoke from the main protX level

\(<\text{segment-statement}\):==

CHAIN { \(<\text{chain-statement}\>\} END generates a sequence of residues.

MOLECule NAME=\(<\text{residue-name}\) NUMBER=\(<\text{integer}\) END generates individual molecules such as waters.

NAME=\(<\text{segment-name}\) specifies the segment name.

\(<\text{chain-statement}\):==

COORDinates { \(<\text{pdb-record}\>\} END reads sequence from a PDB file.

FIRST \(<\text{residue-name}\>

TAIL=\(<\text{patch-character}\>=\(*\text{residue-name}\)* END adds a special patch for the first residue.

LAST \(<\text{residue-name}\>

HEAD=\(<\text{patch-character}\>=\(*\text{residue-name}\)* END adds a special patch for the last residue.

LINK \(<\text{residue-name}\>

HEAD=\(<\text{patch-character}\>=\(*\text{residue-name}\)>

TAIL=\(<\text{patch-character}\>=\(*\text{residue-name}\)* END adds a special linkage patch to the chain database. The statement will automatically connect residue \(i\) to residue \(i+1\); e.g., it creates a peptide linkage. Wildcards are allowed for residue name.

SEQUence { \(<\text{residue-name}\>) \} END takes the sequence as specified. The residue numbers are assigned sequentially, starting with 1.

17.4.2 Example: a polypeptide chain

segment

name="PROT"

chain

link pept head - * tail + * end ! pept patch must be defined

first prop tail + pro end ! special for PRO
17.5 Patching the molecular structure

The patch statement uses a patch residue to add, delete, or modify atoms or bonds. A patch can establish peptide bonds, disulfide bridges, and covalent links to ligands.

17.5.1 Syntax

`PATCh <patch-statement> END` is invoked from the main level of protX.

`<patch-statement>::= <residue-name>`

{ `REFERENCE=NIL` | `<patch-character> = <selection>` } patches the specified selection using the patch residue indicated. The patch character corresponds to the first character in the PRES specification. The specification of NIL implies that in the PRESidue the patch characters are omitted.

17.5.2 Example: a disulfide bridge

```
topology	presidue DISU
group
modify atom 1CB                              charge= 0.19  END
modify atom 1SG type=S charge=-0.19  END
group
modify atom 2CB                              charge= 0.19  END
modify atom 2SG type=S charge=-0.19  END
add bond 1SG 2SG
add angle  1CB 1SG 2SG
add angle  1SG 2SG 2CB
add dihedral  1CA 1CB 1SG 2SG
add dihedral  1CB 1SG 2SG 2CB
add dihedral  1SG 2SG 2CB 2CA
```

end
end
patch
17.6 Deleting atoms

The delete statement removes atoms. It will also delete related bonds, bond angles, or dihedrals.

```plaintext
DELEte { <delete-statement> } END to invoke from the main protX level.
```

- `<delete-statement>`:==
  - `SELEction=<selection>` selects the atoms that are to be deleted.

For example:

```plaintext
delete selection=(resid 1 and name HN) end
```

17.7 Duplicating the Molecular Structure

This statement allows one to duplicate the molecular structure or selected atoms.

```plaintext
DUPLicate { <duplicate-statement> } END to invoke from the main protX level.
```

- `<duplicate-statement>`:==
  - `RESIdue=<residue-name>` specifies the residue name of the duplicated atoms (default: same as original atoms).
  - `SEGId=<segid-name>` specifies the segment name of the duplicated atoms (default: same as original atoms).
  - `SELEction=<selection>` selects the atoms that are to be duplicated.

17.8 Structure statement

The structure statement allows one to read a molecular structure file that has been written previously by the write structure statement.

```plaintext
STRUcture { <structure-statement> } END to invoke from the main protX level.
```

- `<structure-statement>`:==
<psf-records> adds <psf-records> to the molecular structure database.

RESET eliminates the current molecular structure.

For example:

structure ! two structures are read and appended
   @molecule1.psf
   @molecule2.psf
end

17.9 Writing a molecular structure file

The write structure statement writes the current molecular structure to a file, called a PSF for historical reasons:

WRITe STRUcture OUTPut=<filename> END to invoke from the main protX level.
Chapter 18

Energy function

18.1 Empirical Energy Functions

The energy function has the form

$$E_{\text{EMPirical}} = \sum_{p=1}^{N} \left[ w_{BOND}^p E_{\text{BOND}} + w_{\text{ANGL}}^p E_{\text{ANGL}} + w_{\text{DIHE}}^p E_{\text{DIHE}} + w_{\text{IMPR}}^p E_{\text{IMPR}} + w_{\text{VDW}}^p E_{\text{VDW}} + w_{\text{ELEC}}^p E_{\text{ELEC}} \right].$$

(18.1)

to which implicit solvent contributions can be added (see part IV). The sum is carried out over all double selections of atoms (see Section 18.6) with weights $w_n^p$. The default is one double selection involving all atoms with unit weights. In the next sections, the energy terms are described in more detail.

18.2 Bonded terms

The term

$$E_{\text{BOND}} = \sum_{\text{bonds}} k_b (r - r_0)^2$$

(18.2)

describes the covalent bond energy; the sum is carried out over all covalent bonds in the molecular structure selected by the constraints interaction statement.

The term

$$E_{\text{ANGL}} = \sum_{\text{angles}} \left( k_\theta (\theta - \theta_0)^2 + k_{ub} (r_{13} - r_{ub})^2 \right)$$

(18.3)

describes the bond angle energy; the sum is carried out over all bond angles in the molecular structure selected by the constraints interaction statement. The second term in Eq. 18.3 is the Urey-Bradley term, which is used by certain force fields (Burkert and Allinger 1982). The default value for $k_{ub}$ is zero.
The terms 
\[ E_{DIHE} = \sum_{dihedrals} \sum_{i=1,m} \left\{ \begin{array}{ll} k_{\phi_i} (1 + \cos(n\phi_i + \delta_i)) & \text{if } n_i > 0 \\ k_{\phi_i} (\phi_i - \delta_i)^2 & \text{if } n_i = 0 \end{array} \right. \] (18.4) 
\[ E_{IMPR} = \sum_{impropers} \sum_{i=1,m} \left\{ \begin{array}{ll} k_{\phi_i} (1 + \cos(n\phi_i + \delta_i)) & \text{if } n_i > 0 \\ k_{\phi_i} (\phi_i - \delta_i)^2 & \text{if } n_i = 0 \end{array} \right. \] (18.5)
describe the dihedral and improper energy terms. \( \phi_i \) is the actual torsion angle, \( k_{\phi_i} \) are energy constants, \( n_i \) are periodicities, \( m_i \) are multiplicities, and \( \delta_i \) are phase shifts (Section 17.2.1). The specification of multiple dihedral or torsion angles with \( m > 1 \) allows one to carry out a cosine expansion of a torsion potential. Internally, protX stores multiple dihedral or improper angles as multiple instances of the same combination of atoms or atom types.

18.3 Nonbonded energy terms

Three combinations of nbonds options are possible. The first is TRUNcation in combination with CDIE. The second involves a switched van der Waals (VSWItch) and a shifted electrostatic function (SHIFt) in combination with CDIE. The third uses a switched van der Waals function (VSWItch) in combination with a switched electrostatic function (SWITch) and a \( 1/R \) dielectric function (RDIE). All nonbonded energy terms are truncated for atom pairs that are too close to each other (IN-HIbit option in the nonbonded statement, Section 17.2.1). This reduces numerical instabilities.

18.3.1 Van der Waals function

The van der Waals function is given by

\[ f_{VDW}(R) = \left\{ \begin{array}{ll} \frac{A}{R^{12}} - \frac{B}{R^6} = 4\varepsilon((\frac{\sigma}{R})^{12} - (\frac{\sigma}{R})^6)H(R - R_{cut}) & \text{truncation} \\ \left(\frac{A}{R^{12}} - \frac{B}{R^6}\right) SW(R, R_{on}, R_{off}) & \text{switched} \end{array} \right. \] (18.6)

where \( H \) is the heavy-side function and \( SW \) is a switching function. \( SW \) has the form

\[ SW(R, R_{on}, R_{off}) = \left\{ \begin{array}{ll} 0 & \text{if } R > R_{off} \\ \frac{(R^2 - R_{off}^2)(R^2 - R_{on}^2) - 3(R^2 - R_{on}^2)}{R_{off}^2 - R_{on}^2} & \text{if } R_{off} > R > R_{on} \\ 1 & \text{if } R < R_{on} \end{array} \right. \] (18.7)
For both the truncated and the switched option, the van der Waals function is described by a Lennard-Jones potential. The NBON statement (Section 17.2.1) defines $\varepsilon, \sigma$ for the Lennard-Jones potential between identical atom types. Between different atom types, the following combination rule is used:

$$\sigma_{ij} = \frac{\sigma_{ii} + \sigma_{jj}}{2} \quad (18.8)$$

$$\varepsilon_{ij} = \sqrt{\varepsilon_{ii}\varepsilon_{jj}} \quad (18.9)$$

The NBFix statement allows one to deviate from this combination rule.

### 18.3.2 Electrostatic function

The electrostatic function is given by

$$f_{ELEC}(R) = \begin{cases} 
Q_i Q_j \frac{C}{\varepsilon_o R_{\text{heavy}}} (R - R_{\text{cut}}) & \text{for pure truncation} \\
Q_i Q_j \frac{C}{\varepsilon_o R^2} (1 - \frac{R^2}{R_{\text{off}}^2})^2 & \text{for shifted option} \\
Q_i Q_j \frac{C}{\varepsilon_o R^2} \text{SW}(R, R_{\text{on}}, R_{\text{off}}) & \text{for } 1/R \text{ option}
\end{cases} \quad (18.10)$$

### 18.3.3 Intramolecular interactions

The intramolecular interaction energy is the sum of the individual nonbonded interaction energies for pairs of atoms within the current molecular structure:

$$E_{ELEC} = \sum_{i<j} f_{ELEC}(R_{ij}) + e_{14} \sum_{(i,j)\in\{1-4\}} f_{ELEC}(R_{ij}) \quad (18.11)$$

$$E_{VDW} = \sum_{i<j} f_{VDW}(R_{ij}) + \sum_{(i,j)\in\{1-4\}} f_{VDW}(R_{ij}) \quad (18.12)$$

The summation extends over all pairs of atoms that satisfy the cutoff criteria and are selected by the constraints interaction statement.

There are a number of cases where nonbonded interactions must not be computed, e.g., between covalently bonded atoms. Covalently bonded exclusions are automatically generated. In addition, exclusions can be added manually by the EXCLude statement (see Section 17.1.1). The NBXMod statement (see Section 17.2.1) has several options for automatically excluding 1–2, 1–2 and 1–3, and 1–2, 1–3, and 1–4 interactions in the molecule. If NBXMod=±5, electrostatic 1–4 interactions are scaled by $e_{14}$, and the van der Waals interactions use a special 1–4 set of parameters. If NBXMod#±5, 1–4 interactions are treated as normal nonbonded interactions.
18.4 Turning energy terms on or off

The flag statement allows the user to turn energy terms on and off:

\[ \text{FLAGs} \{ \text{flag-statement} \} \text{ END} \] to invoke from the main protX level

\[ \text{flag-statement} := \]

\[ \text{EXCLude} \{ \text{*energy-term*} \} \text{ excludes specified energy-terms.} \]
\[ \text{INCLude} \{ \text{*energy-term*} \} \text{ includes specified energy-terms.} \]

\[ \text{energy-term} := \]

\text{ANGL} specifies bond angle energy (default: on).
\text{BOND} specifies covalent bond energy (default: on).
\text{CDIH} specifies dihedral angle restraints energy (default: off).
\text{DIHE} specifies dihedral angle energy (default: on).
\text{ELEC} specifies intramolecular electrostatic energy (default: on).
\text{HARM} specifies a harmonic energy that restrains the positions of the molecule (default: off).
\text{IMPR} specifies improper dihedral angle (e.g., chirality and planarity) energy (default: on).
\text{PLAN} specifies planarity restraints energy (default: off).
\text{PVDW} specifies symmetry-related van der Waals energy (default: off).
\text{VDW} specifies intramolecular van der Waals energy (default: on).

18.5 Energy statement

\text{ENERgy END} to invoke from the main protX level.

The energy statement performs a single calculation of all energy terms that are turned on. The atomic forces are also computed and stored in arrays DX, DY, and DZ. Upon completion of the energy calculation, symbols are declared that contain the computed energy terms. The overall energy (Eq. 18.1) is stored in the symbol $ENER$; the rms gradient is stored in $GRAD$. 
18.6 Energy calculation between selected atoms

The constraints interaction statement tells protX to compute the energy only between two selected sets of atoms, called a double selection. For two-point energy terms (such as covalent bonds and nonbonded interactions), the energy is computed if one atom of the bond belongs to the first selection and the other belongs to the second selection. For three-point terms (such as angles) and four-point terms (such as dihedrals), the energy is computed if at least one atom belongs to the first selection, at least one other atom belongs to the second selection, and all atoms of the three-point or four-point term belong to at least one selection. The statement can be issued several times, defining several double selections. In that case, the total energy and the total forces are obtained by summing over the different double selections. In addition, when a double selection is defined, the user may attribute a weight to each individual energy term (bonds, angles, etc.). A constraints statement will automatically erase all previous double selections.

\[
\text{constraints}
\] \[
\text{interaction}= ( \text{segid "A"} ) ( \text{segid "A"} )
\]
\[
\text{interaction}= ( \text{segid "B"} ) ( \text{segid "B"} )
\]
end

18.6.1 Syntax

CONStraints \{ < constraints-interaction-statement > \} END to invoke from the main protX level

\[<\text{constraints-interaction-statement}>::=\]
\[\text{INTERation}= <\text{selection}> <\text{selection}> [ \{ <\text{weight-statement}> \} ]\]

The default is a single double selection involving all atoms of the molecular structure.

\[<\text{weight-statement}>::=\]
\[\text{WEIghts} \{ <\text{*energy-term*}> <\text{real}> \} END\]

applies the weight (real) to the specified energy term.

This example below excludes the intrasegment angles, dihedrals, impropers, and nonbonded terms:

CONStraints
\[
\text{INTERaction} ( \text{segid a} ) ( \text{segid b} ) \text{WEIGhts} * 1. END
\]
\[
\text{INTERaction} ( \text{segid a} ) ( \text{segid a} ) \text{WEIGhts} * 0. \text{bonds} 1. END
\]
INTEraction ( segid b ) ( segid b ) WEIGHTs * 0. bonds 1. END
END
Chapter 19

Geometric and energetic analysis

19.1 Analysis of conformational energy terms

The print statement provides information about selected bonds, angles, dihedrals,
impropers. The pick statement allows one to pick specific energy terms. Both assign
results to $RESULT.

**PRINT** `<print-statement>` to invoke from the main protX level

 `<print-statement>::= [THRESHold=<real>] <print-objects>` prints objects. (default: THRESHold=0)

 `<print-objects>::=`

  ANGLe lists bond angles that deviate by more than THRESHold
  BOND lists bond lengths that deviate by more than THRESHold
  CDIHEdral lists dihedral restraints that deviate by more than THRESHold
  DIHEDrals lists dihedrals that deviate by more than THRESHold
  IMPRopers lists impropers that deviate by more than THRESHold

**PICK** `<pick-statement>` to invoke from the main protX level

 `<pick-statement>::=`

  ANGLE <selection> <selection> <selection> <property>
  BOND <selection> <selection> <property>
  DIHEDrals <selection> <selection> <selection> <selection> <property>
  IMPRoper <selection> <selection> <selection> <selection> <property>

 `<property>::= ENERgy|GEOMetry`
For example, to print bonds that deviate from ideal geometries, then extract specific distances:

```
print threshold=0.1 bonds
print threshold=10.0 angles
cons inter (resid 40) (resid 40) end ! residue 40 only
print threshold=0.1 bonds
pick bond ! get geometry of a CO bond
(resid 1 and name c) (resid 1 and name o) geometry
end

pick bond ! get distance between two atoms
(resid 5 and name nz) (resid 1 and name o) geometry
end
```

To extract the angle among three arbitrary atoms (not necessarily bonded):

```
pick angle (resid 1 and name c)(resid 32 and name n)(resid 5 and name ca) geom
```

### 19.2 Analysis of the nonbonded energy terms

The distance statement allows one to analyze nonbonded interactions or contacts. Selected parts of the nonbonded list may be printed by specifying an upper and lower cutoff and atom selections. One can also produce a distance matrix.

**DISTance** { <distance-statement> } to invoke from the main protX level

```
<distance-statement>:==

CUTOFF=<real> upper distance cutoff: distances less than CUTOFF and less than the list cutoff (CUTNB) are analyzed.

CUTON=<real> is a lower distance cutoff.

DISPosition=<MATRix|PRInt|RMSD> specifies how distances will be stored or printed. RMSD stores the minimum distance for each atom in the 1st selection to all atoms in the 2nd selection in the RMSD array. PRInt writes all selected nonbonded distances to standard output. MATRix stores all selected, nonbonded distances in a matrix and writes the matrix to the specified output file.

FROM=<selection> first atom selection (default: (ALL)).

OUTPut=<filename> specifies a file for the distance matrix.

TO=<selection> second atom selection.
For example,

```plaintext
parameter nbonds cutnb=20. end
distance from=(resid 10) to=(resid 70) cuton=0. cutoff=20. end
```
Chapter 20

Cartesian coordinates

20.1 Coordinate statement

The coordinate statement is used to read and manipulate coordinates, such as rotation, translation, or fitting to a comparison coordinate set.

COORdinates <coordinate-statement> END to invoke from the main protX level

<coordinate-statement>:==

COPY [SELEction=<selection>] copies main coordinates into comparison set XCOMP, YCOMP, ZCOMP

FIT { [SELEction=<selection>] [MASS=<logical>] [LSQ=<logical>] } rotates (if LSQ) and translates all main coordinates to fit the selected comparison atoms. The Euler angles and translation vector are stored in $THETA1, $THETA2, $THETA3, $X, $Y, $Z.

INITialize { [SELEction=<selection>] } initializes main coordinates.

ORIEnt { [SELEction=<selection>] [MASS=<logical>] [LSQ=<logical>] } rotates (if LSQ) and translates all coordinates so that the principal axes of the selected atoms correspond to x,y,z.

RGYRation { [SELEction=<selection>] [MASS=<logical>] [FACT=<real>] } computes radius of gyration and declares the symbols $RG (radius of gyration), $XCM, $YCM, $ZCM (center of mass).

RMS { [SELEction <selection>] [MASS=<logical>] } computes the rms difference for selected atoms between the main and comparison set.

ROTAtE { [SELEction=<selection>] [CENTer=<3d-vector>] <matrix> } rotates selected atoms around the specified rotation center
CHAPTER 20. CARTESIAN COORDINATES

The rotation matrix is specified through the matrix statement.

\texttt{SWAP \{ [SELEction=\textless selection\textgreater] \} \ exchanges \ main \ and \ comparison \ coordinates.}

\texttt{TRANslate \{ [SELE=\textless selection\textgreater] VECTor=\textless 3d-vector\textgreater \ [DISTance=\textless real\textgreater]\}
\ transmits \ selected \ atoms.}

\texttt{COOR <coordinate-read-statement> END \ reads \ coordinates.}

\texttt{<coordinate-read-statement>:== [DISPosition= \text{COMP}arison \ | \ MAIN
| \ REFERence \] [SELE=\textless selection\textgreater]\{ <pdb-record> \} \ reads \ into \ the \ main \ (X,Y,Z,B,Q), \ comparison \ (XCOMP,YCOMP,ZCOMP,BCOMP,QCOMP), \ or \ reference \ (REFX, REFY, REFZ, HARM, HARM) \ arrays.}

For example, to fit to a comparison structure using $C_\alpha$ atoms, then compute the rms deviation:

\texttt{coor fit sele=(name ca) end}
\texttt{coor rms sele=(name ca or name n or name c) end}
\texttt{evaluate ($rmsdev =$result)}
\texttt{vector show (b) ! display rms differences above 1 Angstrom}
\texttt{\hspace{1cm} (attrib b > 1.0 and (name ca or name n or name c))}

20.2 Rotamer implementation in protX

When Proteus prepares the system, it places each possible rotamer at each residue position. protX uses the concept of “resclass”, which identifies a residue by its resid, resname, and segid. These quantities are available in the PDB format and within protX. A “model” is defined to be a coordinate set of a resclass. One resclass can have multiple models, which can be thought of as different rotamers. The resclass is hidden to the user, who manipulates only models. A model can be declared in the ATOM statement of a PDB file, just before the segid field (see below). Models can be read in two ways. The command

\texttt{coor disp=model @file.pdb}

adds each model found in file.pdb to memory. A model number is read from PDB columns 67-71. It represents the model number among those associated with the given resclass. The command

\texttt{coor disp=model push=true @file.pdb}
adds a single model to a resclass. The model number is not read but generated by incrementing the last model number. Models can be copied:

```plaintext
coor copy from=A to=B idx=i=j end
```

where A, B can be any of main, comp, xref or model. The specification idx=i=j can be omitted. By default, when using from=model, idx is 1 and the new model number is generated automatically. The command:

```plaintext
write coor sele=(resid $1 and resn $aa1) from=model output=new.pdb end
```

writes all the models of the selected resclasses to a PDB file new.pdb. To output only one model:

```plaintext
write coor from=model idx=i output=new.pdb end
```

In the following PDB lines, the model number is indicated just before the segid:

```
ATOM  339  N  GLY   3  9  -3.933  5.444  16.117   1.00   1.00  13  A
ATOM  341  CA  GLY   39  -4.479  6.276  17.176   1.00   1.00  13  A
ATOM  342   C  GLY   39  -3.682  7.552  17.389   1.00   1.00  13  A
ATOM  343   O  GLY   39  -4.251  8.617  17.614   1.00   1.00  13  A
ATOM 1044  OD1  ASP  111  -13.801 -5.521  -3.500   1.00   0.00  14  A
ATOM 1045  OD2  ASP  111  -14.043 -4.328  -5.339   1.00   0.00  14  A
ATOM 1046   C  ASP  111  -12.244  -8.798  -5.753   1.00   0.00  14  A
ATOM 1047    O  ASP  111  -11.038  -9.008  -5.762   1.00   0.00  14  A
ATOM 1048   N  LYS  112  -13.097  -9.470  -6.515   1.00   0.00  14  A
ATOM 1050   CA  LYS  112  -12.655 -10.531  -7.415   1.00   0.00  14  A
ATOM 1051  CB  LYS  112  -13.852 -11.135  -8.142   1.00   0.00  14  A
```

### 20.3 Write coordinate statement

The write coordinate statement writes the current coordinates to a specified file.

```plaintext
WRIte COOR { <write-coordinate-statement> } END to invoke from main protX level

<write-coordinate-statement>:=

FROM= MAIN | COMP | REFE (default: MAIN).
OUTPUT=<filename> specifies the output filename.
SELE=<selection> writes selected coordinates (default: (ALL)).
```
20.4 Building hydrogen positions

The hbuild statement builds the selected hydrogens (Brünger and Karplus 1988). It performs local energy minimization in cases where the placement of the hydrogens is not unique.

\texttt{HBUILD \{ <hbuild-statement> \} END} to invoke from the main protX level

\texttt{<hbuild-statement>:==}

\hspace{1cm} \texttt{ACCEptor=<selection>} selects atoms that should be perceived as acceptors for hydrogen bonds involving waters (default: atoms that have an explicit ACCEptor assignment; see Section 17.1.1).

\hspace{1cm} \texttt{PHIStep=<real>} specifies the step size for the dihedral angle search (default: 10\degree).

\hspace{1cm} \texttt{PRINT} is a flag that provides information during the local minimization.

\hspace{1cm} \texttt{SELEction=<selection>} specifies a selection of atoms to build.
Chapter 21

Coordinate restraints and constraints

21.1 Harmonic coordinate restraints

A point restraint energy can be defined:

\[ E_{HARM} = \sum_{\text{atoms}} h_i (r_i - r_{i}^{\text{ref}})^e \]  \hspace{1cm} (21.1)

where the sum extends over all atoms, \( h_i \) are individual weights, \( r_i \) are the main coordinates, \( r_{i}^{\text{ref}} \) are reference coordinates, and \( e \) is an exponent. The weights \( h_i \) are in the atom array HARM and can be assigned using the vector statement. The exponent \( e \) is set by the restraints harmonic statement.

A planar restraint can be defined:

\[ E_{HARM} = \sum_{\text{atoms with } h_i < 0} (-h_i) \left( \frac{\vec{n}}{|\vec{n}|} \cdot (\vec{r}_i - \vec{r}_{i}^{\text{ref}}) \right)^e \]  \hspace{1cm} (21.2)

where the sum extends over all atoms with negative weights \( h_i \). A nonzero normal vector \( \vec{n} \) has to be specified using the restraints harmonic statement. Note that plane restraints are computed only for atoms with \( h_i < 0 \); otherwise point restraints are applied, allowing simultaneous use of point and planar restraints.

\texttt{restraints HARMonic \{ <restraints-harmonic-statement> \} END} to invoke from main level. This statement automatically turns on the HARM energy flag (Section 18.4).

\texttt{<restraints-harmonic-statement>:=}

\texttt{EXPO}onent=\texttt{<integer>} specifies the exponent \( e \) (default: 2).

153
\textbf{NORMal=\langle vector\rangle} specifies the normal vector \( \vec{n} \). If \( \vec{n} \neq (0,0,0) \), plane restraints are enabled (default: \( (0 \ 0 \ 0) \)).

For example:

coordinates @file1
coordinates disp=reference @file2
vector do (harm=20.0 ) (name ca)
vector do (harm=0.0 ) (not name c )
restraints harmonic exponent=2 end
flags include harm end

\section*{21.2 Dihedral restraints}

A dihedral restraint can be defined: energy \( E_{CDIH} \) is given by

\[ E_{CDIH} = S \sum C \text{well}(\text{modulo}_{2\pi}(\phi - \phi_o), \Delta \phi)^{ed} \]  

where the sum extends over all restrained dihedral angles, \( S \) is a weight, and the flat-bottom potential \( \text{well}(a,b) \) is given by

\[ \text{well}(a,b) = \begin{cases} 
  a - b & \text{if } a > b \\
  0 & \text{if } -b < a < b \\
  a + b & \text{if } a < -b 
\end{cases} \]  

The constant \( C \), the angle range \( \Delta \phi \), the angle centroid \( \phi_o \), and the exponent \( ed \) are specified in the restraints dihedral statement.

\textbf{RESTraints DIHEdral \{ \langle restraints-dihedral-statement\rangle \} END} to invoke from the main level. Automatically turns on the CDIH energy flag (Section \ref{18.4}).

\textbf{<restraints-dihedral-statement>:==}

\textbf{ASSIgn} <selection> <selection> <selection> <selection>
\textbf{<real>} <real> <real> <integer> adds a new dihedral restraint
The four selections have to be unique (one atom each, not necessarily bonded). The first \textbf{<real>} is the energy constant \( C \) the second specifies the target angle, the third specifies the allowed range around the target.

\textbf{NASSign=<integer>} (required) specifies the maximum expected number of assignments (default: 400).

\textbf{RESEt} erases the restraints-dihedral database.
SCALe specifies the overall weight $S$.

For example:

```plaintext
restraints dihedral nassign=300 scale=1.0
assign (resid 1 and name ca) (resid 10 and name cb)
   (resid 4 and name n) (resid 8 and name sg) 20.0 55.0 0.0 2
assign (resid 3 and name hg) (resid 5 and name o)
   (resid 2 and name cb) (resid 1 and name cg) 20.0 170.0 0.0 2
end
flags include cdih end
```

### 21.3 Planarity restraints

The restraints planarity statement defines an effective energy term $E_{PLAN}$ that penalizes out-of-plane conformations of selected atoms:

$$E_{PLAN} = \sum_{g \in \text{groups}} w_{plan} \sum_{i \in g} g_i^2$$  \hspace{1cm} (21.5)

where the first sum is over all defined groups of planar atoms, the second sum is over all atoms $i$ within each group, and $g_i$ is the orthogonal distance of $i$ from the plane defined by all atoms of the group (Schomaker et al. 1959).

RESTraints PLANar { <restraints-planar-statement> } END to invoke from the main level

<restraints-planar-statement>::=

```plaintext
GROUP { <restraints-plane-group-statement> } END adds a new group to the planar restraints database. More than three atoms per group need to be defined.

INITialize erases the current planar restraints database.
```

<restraints-plane-group-statement>::=

```plaintext
SELECTION=<<selection>> defines the group of atoms.
WEIGHT=<<real>> specifies a weight (default: 300.0 kcal mole$^{-1}$ Å$^{-2}$).
```

### 21.4 Fixing atomic positions

Atomic positions can be fixed during minimization or molecular dynamics.
21.4.1 Syntax

**CONStraints FIX** `<constraints-fix-statement>` **END** to invoke from main level

`<constraints-fix-statement>`::<=

`<selection>` selects atoms to fix.

The following example fixes $C^\alpha$ carbon atoms:

`constraints fix=(name ca) end`

### 21.5 Fixing distances with SHAKE

The SHAKE method (Ryckaert, Ciccotti, and Berendsen 1977) constrains distances between atoms to reference values. The shake statement is used to set up the database of constraints.

**SHAKe** `{ `<shake-statement>` } **END** to invoke from the main level

`<shake-statement>`::<=

**ANGLE** `<selection>` `<selection>` `<selection>` adds new SHAKE constraints. For parameter-based constraints (REFErence=PARAmeter), type-based parameters will be used.

**BOND** `<selection>` `<selection>` adds new SHAKE constraints.

**MOLEcule** `<selection>` adds new SHAKE constraints. Normally used for small molecules like water.

**MXITerations**=`<integer>` specifies the maximum number of SHAKE iterations (default: 500).

**NCONstraints**=`<integer>` allocates space for SHAKE constraints (default: 4000).

**REFErence**= **COORdinates** | **PARAmeters** determines whether the reference distances come from the coordinates or the parameters (default: COORdinate).

**RESET** erases the current SHAKE database.

**TOLERance**=`<real>` specifies the deviation at which iterations are terminated (default: 1.0e-05).
Chapter 22

Conjugate gradient energy minimization

The minimization is started from the atom properties X,Y,Z, and the minimized coordinates are returned in X,Y,Z. SHAKE constraints are possible (cf. Section 21.5). The final energy and gradient are stored in the symbols $ENER and $GRAD.

\text{MINImize POWEll} \{ <\text{minimize-powell-statement}> \} \text{ END} to invoke from main level

\text{<minimize-powell-statement>} ::= 

\hspace{1cm} \text{DROP=}<\text{real}> gives the expected initial drop in energy (default: 0.001). Values between 10 and 100 work best.

\hspace{1cm} \text{NPRInt=}<\text{integer}> is the frequency of the energy printout (default: 1).

\hspace{1cm} \text{NSTEp=}<\text{integer}> is the maximum number of minimization cycles (default: 500).

\hspace{1cm} \text{TOLGradient=}<\text{real}> minimization stops when the gradient norm reaches this value (default: 0.0001).
Chapter 23

Molecular dynamics

Molecular dynamics capabilities are described in the Xplor manual.
List of protX statements

Below are the application statements accessible from the main protX level:

<application-statement>::=
CONStraints FIX <constraints-fix-statement> END
CONStraints { INTER <constraints-interaction-statement> } END
COORdinate <coordinate-statement> END
DELEte { <delete-statement> } END
DISTance { <distance-statement> } END
DUPLicate { <duplicate-statement> } END
DYNAmics MERGe { <dynamics-merge-statement> } END
DYNAmics VERLet { <dynamics-Verlet-statement> } END
ENERgy { <energy-statement> } END
FLAGs { <flag-statement> } END
HBUIld { <hbuild-statement> } END
MINImize POWEll { <minimize-powell-statement> } END
MINImize RIGId { <minimize-rigid-statement> } END
NOE { < noe-statement> } END
PARAmeter { <parameter-statement> } END
PATCh <patch-statement> END
PICK <pick-statement>
PRINt <print-statement>
READ TRAJectory { <read-trajectory-statement> } END
RESTRaints DIHE { <restraints-dihedral-statement> } END
RESTRaints HARM { <restraints-harmonic-statement> } END
RESTRaints PLANar { <restraints-planar-statement> } END
SEGMen { <segment-statement> } END
SHAKE { <shake-statement> } END
STRUcture { <structure-statement> } END
SURFace { <surface-statement> } END
TOPOlogy { <topology-statement> } END
VECThr { <vector-statement> }
WRITe COORdinates { <write-coordinates-statement> } END
WRITe STRUcture { <write-structure-statement> } END
WRITe TRAJectory { <write-trajectory-statement> } END
Bibliography


