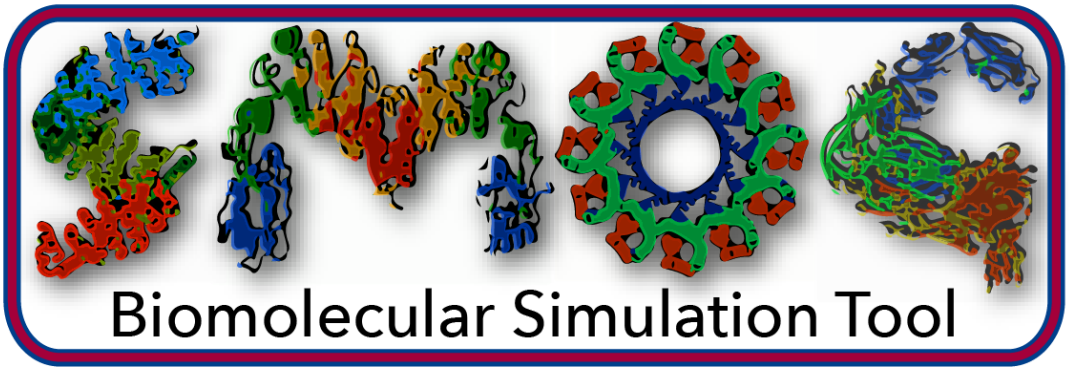
SMOG

Version 2.6-beta
User's Manual

September 12, 2026

Since this is a dynamic document that is continually updated
as the code is modified, it is possible that some elements
did not build properly, or may have errors.

Northeastern University • Rice University



Authors:
Jeffrey Noel, Mariana Levi, Antonio Oliveira, Vinícius Contessoto,
Mohit Raghunathan, Heiko Lammert, Ryan Hayes,
José Onuchic, Paul Whitford
info@smog-server.org

Should you read this manual?

If you would only like to use the basic functionality of SMOG 2 (i.e. the standard supported models), then you may find that the `README` file associated with the distribution provides all the information you need. This manual provides a more detailed description of the basic usage guidelines, in addition to advanced usage information and detailed descriptions of the underlying methodologies/models. For basic users, if the `README` is not sufficient, then Chapters 1, 2, 3 and 4 will help you get started. For more advanced users, who may wish to modify structure-based models (e.g. extending to new residue types, ligands, electrostatics, etc), then consulting Chapters 5, 6, 7 and 8 will be necessary. We additionally provide appendices that have technical details that may be of interest to some users. While we try to provide all pertinent information here, don't hesitate to contact us for clarification.

SMOG 2, and all associated files, are distributed free of charge, made available under the GNU General Public License.

Contents

List of Tables v

1 Introduction 1
- 1.1 What are Structure-Based Models? . . . 1
- 1.2 What does SMOG 2 do? . . . 2

2 Installation 3
- 2.1 Container-based usage . . . 3
  - 2.1.1 Docker container . . . 3
  - 2.1.2 Singularity container . . . 4
- 2.2 Manual configuration (non-container-based usage) . . . 5
  - 2.2.1 Prerequisites . . . 5
  - 2.2.2 Final configuration/install steps . . . 5
  - 2.2.3 Verify your SMOG configuration . . . 6

3 Using SMOG 2 9
- 3.1 Preparing the input PDB file . . . 9
  - 3.1.1 PDB file format . . . 9
  - 3.1.2 Preprocessing the PDB file . . . 10
- 3.2 Generating a Structure-Based Model . . . 10
  - 3.2.1 Interactive force field selection . . . 11
  - 3.2.2 Default All-Atom Model . . . 11
  - 3.2.3 Default C_α model . . . 12
  - 3.2.4 Default Gaussian contact potential models . . . 13
  - 3.2.5 Using non-default models . . . 13
  - 3.2.6 User-provided contact map . . . 13
- 3.3 Input options . . . 14

4 Performing a simulation, or calculation 16
- 4.1 Using OpenMM . . . 16
  - 4.1.1 Native OpenMM support for SMOG models . . . 16
  - 4.1.2 OpenSMOG module for OpenMM . . . 17
    - 4.1.2.1 Basic usage . . . 17
    - 4.1.2.2 Advanced usage (User-defined Custom Potentials) . . . 17
- 4.2 Using Gromacs 4.5 or 4.6 . . . 22
  - 4.2.1 All-Atom Model . . . 22
  - 4.2.2 C_α Model . . . 23
  - 4.2.3 Examples . . . 24

4.2.4 Note on Domain Decomposition . . . 24
4.3 Using Gromacs 5 . . . 24
4.3.1 Examples . . . 25
4.4 Using Gromacs 2020 . . . 25
4.4.1 Examples . . . 25
4.5 Using NAMD . . . 25
4.6 Using LAMMPS . . . 25
4.7 Discrete Path Sampling (only available in Beta version) . . . 26

5 SMOG Tools **29**
5.1 `smog_adjustPDB` . . . 29
5.2 `smog_editgro` . . . 32
5.3 `smog_extract` . . . 33
5.4 `smog_ions` . . . 33
5.5 `smog_modifyXML` . . . 34
5.6 `smog_scale-energies` . . . 35
5.7 `smog_tablegen` . . . 37
5.8 `SCM.jar` . . . 38
5.8.1 Corrected algorithm . . . 41
5.9 `WHAM.jar` . . . 42
5.9.1 Configuration file . . . 43
5.9.1.1 Example 1: Combining constant temperature runs . . . 43

6 Template-Based Approach **47**
6.1 Introduction to templates . . . 47
6.2 SMOG 2 Templates . . . 47
6.2.1 Biomolecular Information File (.bif) . . . 48
6.2.1.1 Residues . . . 48
6.2.1.2 Connections . . . 50
6.2.2 Setting Information File (.sif) . . . 51
6.2.2.1 Functions . . . 51
6.2.2.2 Declaring a contact type . . . 54
6.2.2.3 Defining the contact map . . . 54
6.2.2.4 Group Settings . . . 55
6.2.3 Bond File (.b) . . . 56
6.2.3.1 How SMOG matches dihedral parameters to specific dihedral angles . . . 59
6.2.4 Nonbond File (.nb) . . . 59
6.2.5 Extras file . . . 61
6.2.6 ions.def file . . . 61

7 Adding a new residue **62**
7.1 Step 1 - Examine the molecular structure . . . 62
7.2 Step 2 - Create a new All-Atom template directory . . . 63
7.3 Step 3 - Define a new residue . . . 63
7.3.1 Place the new residue tag in the .bif file . . . 63
7.3.2 List all of the atoms in the residue . . . 64

7.3.3 List all of the bonds . . . 65
7.3.4 List the improper dihedrals . . . 68
7.4 Step 4 - Define a non-bonded group in the .nb file . . . 69

8 Additional supported interaction types 71
8.1 Adding non-standard bonds between specific atoms . . . 71
8.2 Including atom-type-specific parameters in a SMOG model . . . 72
8.3 Including perfectly free angles, dihedrals and contacts . . . 73
8.4 Adding electrostatics . . . 74

A Energetic Description of the Distributed Models 76
A.1 The All-Atom model . . . 76
A.2 The C_α model . . . 78
A.3 Gaussian contact potential (+gaussian templates) . . . 78
A.3.1 templates/SBM_AA+gaussian . . . 79
A.3.2 templates/SBM_calpha+gaussian . . . 79
A.3.3 Dual-basin Gaussian potential . . . 80
A.3.4 Downloading the source code extensions . . . 80
A.3.4.1 OpenSMOG . . . 80
A.3.4.2 Gromacs . . . 80
A.3.4.3 NAMD . . . 80
A.3.5 Including Gaussian potentials in the topology files . . . 80
A.3.5.1 Gromacs . . . 80
A.3.5.2 NAMD . . . 81
A.4 Reduced units . . . 81

B Installing Perl Modules using CPAN 83
B.1 Introduction . . . 83
B.2 Installing CPAN . . . 83
B.3 Upgrading your perl version using CPAN . . . 84
B.4 Example installation of a Perl module . . . 85
B.5 Troubleshooting tips . . . 85

C FAQs and Tips 87

D Acknowledgements 89

Bibliography 90

List of Tables

3.1 Flags supported by SMOG v2.6-beta 15

5.1 Flags supported by `smog_adjustPDB` 32
5.2 Flags supported by `smog_editgro` 33
5.3 Flags supported by `smog_extract` 34
5.4 Flags supported by `smog_ions` 35
5.5 Flags supported by `smog_modifyXML` 36
5.6 Flags supported by `smog_scale-energies` 36
5.7 Flags supported by `smog_tablegen` 37

6.1 Descriptions of SMOG 2 template files 48
6.2 Potential energy functions available in SMOG 2 53

Chapter 1

Introduction

1.1 What are Structure-Based Models?

Structure-based models (i.e. SBMs, or SMOG models) define a particular known conformation (or a subset of interactions defined by the structure) as a potential energy minimum. With this being the only requirement, there is an endless number of ways in which one may construct a structure-based model. For example, one may build protein-specific and RNA-specific variants, the resolution of the model can be varied[1, 2], multiple minima may be included[3, 4], the degree to which non-native interactions are stabilizing can be adjusted, and electrostatics (explicit, or implicit ions) can be included[5]. The utility of these models is equally broad, where they may be applied for understanding dynamics, or for structural modeling objectives, as discussed elsewhere[6]. With such flexibility, this general class of models can be tailored to ask specific questions about biomolecular processes. In the present document, we describe a set of computational tools that allows one to use previously-developed structure-based models, as well as design and implement new variations that are suited for your specific needs.

In the simplest form, a structure-based model defines a single configuration as the global potential energy minimum, where all intra- and inter-molecular interactions are assigned minima that correspond to that structure. This fully native-centric variant of the model is colloquially referred to as a "vanilla" structure-based model. In terms of the energy landscapes of biomolecules, these vanilla models represent an energetically unfrustrated landscape[7, 8]. Since biomolecular landscapes possess some degree of energetic roughness, it is often desirable to extend structure-based models to include both native and non-native interactions. As such, in the SMOG 2 software package, we provide two energetically unfrustrated models by default, upon which additional interactions may be added by the user. Specifically, this distribution provides the coarse-grained C_α

structure-based model for proteins, as developed by Clementi et al. [1] and the all-atom structure-based model, as developed by Whitford et al. [2]. While the C_α model is only defined for proteins, the all-atom model supports proteins, RNA, DNA and some ligands. Non-default force field parameters may be downloaded from the smog-server force field repository. Since there has been a number of extensions in the all-atom model over the last several years, a complete description of the energetic parameters is given in Appendix A.

1.2 What does SMOG 2 do?

SMOG 2 is a software package designed to allow the user to start with a structure of a biomolecule (i.e. a PDB file) and construct a structure-based model, which is then simulated using Gromacs [9], NAMD [10], openMM, or LAMMPS. In addition, the OpenSMOG libraries provide an interface between SMOG 2 and OpenMM that allows for a much wider range of models to be developed. We previously implemented an online server (SMOG 1) that was capable of providing the vanilla flavor of structure-based models, along with a few adjustable parameters. SMOG 2 is a nearly complete rewrite of the original software package, and it provides four major advantages over its predecessor:

- Extensibility – One may add new residue and molecule types without source-code modifications.
- Portability – By building force field definitions on generally-defined XML-formatted files, researchers may easily distribute and share new SMOG model variants. We encourage users to make their models publicly available through the SMOG 2 Force Field Repository.
- Generalizability – Every energetic parameter may be varied, and additional energetic interactions (even non-native) may be included.
- Multi-resolution capabilities – Any level of structural resolution may be implemented, as well as multi-resolution variations.

It is important to note that, in order to adjust the general definition of a class of SMOG models, one simply needs to introduce changes to the XML template files. The templates are not statically-linked to SMOG 2, which allows any user to easily choose from a library of models at runtime.

Chapter 2

Installation

The SMOG 2 software is available as a direct source-code download, or as a preconfigured container. If one chooses to download the code, or use the beta/git version, it is necessary to configure a few settings and ensure that appropriate modules are available at runtime. Below are instructions on how to configure SMOG 2 properly on your local machine (source-code download), as well as information on the preconfigured container options.

2.1 Container-based usage

The easiest and fastest way to get started with SMOG 2 is to use a container. As described at the Docker website: A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

We provide containerized forms of SMOG 2 via Docker and Singularity images. Generally, if you are using a personal computer, you may find Docker to be easier. However, if you plan to run SMOG 2 on a shared resource, Singularity may be preferred. The SMOG 2 distribution/repo also provides detailed instructions for generating containers with Docker and Singularity, so that users can build customized containers, if needed.

2.1.1 Docker container

If Docker is running on your computer, the following command will pull the container with the most recently-released version of SMOG 2 (change the tag for other versions, such as "gitversion", or "v2.4.4"):

```
> docker pull smogserver/smog2:stable
```

The container can then be launched with:

```
> docker run -it --rm smogserver/smog2:stable
```

You are initially placed into the directory /workdir inside the container at a bash prompt. Note that the container is essentially a stripped down Ubuntu Linux. `vi` and `nano` text editors are available. All SMOG 2 executables are available in the $PATH.

IMPORTANT: You will want to connect the container to the files on your computer, e.g. to load PDB files and to write output files. This is done with the `-v` switch, which connects (mounts) a directory on your file system to a directory inside the container. This can be done however you like, but we recommend the following:

```
> docker run -it --rm -v $(pwd):/workdir smogserver/smog2:stable
```

$pwd refers to your present working directory. `/workdir` is an arbitrary directory that we have created inside of the container. When inside the container, it should thus effectively start you in your present working directory with a bash prompt. Note that in this case, even though the directory is named `/workdir`, it is in fact the directory from which you launched docker, so editing and deleting files will actually change the files on your machine! Make sure all the files you need are available in `$pwd` and its subdirectories, as these files will be available to the container in `/workdir` and its subdirectories. The container is unable to interact with any directories above `$pwd`.

2.1.2 Singularity container

If Singularity is installed on your computer, the following command will pull the container for the most recent released version of SMOG 2:

```
> singularity pull --arch amd64 library://smog-server/library/smog2:stable
```

If you want to use a different version, then change "stable" to the desired version (e.g. gitversion, v2.4.4, etc). The above command will download the container in the form of a Singularity Image Format file: smog2_stable.sif

You may find that a specific container is available in Docker, but not Singularity. In that case, you can pull and convert the Docker container into a Singularity sif file with the command (this example simply pulls the stable release):

```
> singularity pull smog2_stable.sif docker://smogserver/smog2:stable
```

The container can then be launched with:

```
> singularity shell smog2_stable.sif
```

At this point, the only visible difference will be that the command prompt says "singularity". However, SMOG 2 and all SMOG Tools will be available in your path. Just type "smog2" to get going!

2.2 Manual configuration (non-container-based usage)

2.2.1 Prerequisites

SMOG 2 runs on all Unix-like operating systems. The prerequisites for SMOG 2 are

Perl Programming Language
Perl Data Language (PDL),

as well as the following modules, which are available through the Perl module managing utility CPAN, conda, apt-get (Ubuntu) or manual installation:

```
XML::Simple
XML::SAX::ParserFactory
XML::Validator::Schema
XML::LibXML
Exporter
PDL
Getopt::Long
Scalar::Util
```

Finally, your machine must have `Java Runtime Environment v1.7` or greater.

We have found that using Conda, or apt-get to install required libraries is typically more straightforward than relying only on CPAN. If you would like to use Conda, or apt-get, see the README file for details.

2.2.2 Final configuration/install steps

Note: configuration steps changed after SMOG v2.4.4! If you are using v2.4.4, or an earlier version, please find the configure instructions in the associated manual.

Before running SMOG 2, you must configure it on your local machine. This is accomplished through a one-step (sometimes two-) process:

(**common usage**) Configure with:
```
> source configure.smog2
```

This will set the required environment variables for your current session. Among other things, this will add the smog bin to your `PATH` and perform some basic health checks. Sourcing only sets up your environment for the current session. If you want to use SMOG 2 frequently, you should put the bin directory in your `PATH`. When you run configure.smog2, it will tell you exactly how to update your path for persistent usage.

(**give Perl version**) When sourcing configure.smog2, the script will search for an available version of Perl, and the environment will be set using that version. However, this is not guaranteed to find the version that you intend. If you want to use a different version of Perl, you have to set the `perl4smog` variable. On most linux systems, the default location of Perl is `"/usr/bin/perl"`, whereas on OSX it is typically `"/opt/local/bin/perl"`. Setting `perl4smog` can be accomplished in two ways.

First, you may export the values:
```
> export perl4smog=<give value>
> source configure.smog2
```

The second option is to set the variable:

```
> perl4smog=<give value> source configure.smog2
```

(**Install in a specific location**) Instead of configuring SMOG 2 in the current directory, one may also have the contents copied to a desired directory, configure the program and then write-protect the files. This is desirable when using a shared resource. It is also a good idea if you think you are prone to accidentally editing/corrupting code. To specify the install location, set the variable `smog_install_dir`.

As a note, we have found that some Linux distributions require that you replace `source` with `bash`.

2.2.3 Verify your SMOG configuration

If SMOG is properly configured, then you will be able to run SMOG 2 with the following command:

```
> smog2
```

and you will be greeted with a message that looks similar to:

```
*****************************************************************************************
******* ******* ******* ******* ******* SMOG v2.0 ******* ******* ******* ******* *******
             Thank you for using the Structure-based Model (SMOG) software

    This package is the product of contributions from a number of people, including:
                    Jeffrey Noel, Mariana Levi, Mohit Raghunathan,
                       Ryan Hayes, Jose Onuchic & Paul Whitford

                    Copyright (c) 2015, The SMOG development team at
                       Rice University and Northeastern University

                SMOG v2.0 & Shadow are available at http://smog-server.org

                          Direct questions to: info@smog-server.org

*****************************************************************************************
```

In addition to verifying that SMOG 2 will launch, it is highly recommended that you also run the full set of test scripts, which are in the SMOG-CHECK subdirectory of the distribution.

SMOG-CHECK contains a single script that will launch tests of smog2, smog tools and the Shadow Contact Map program (SCM). When everything works well, performing the checks is as easy as issuing a single command (assuming `configure.smog2` was sourced and smog2 is in your path):

```
>./checkall
```

If all tests are passed, then you are ready to go! No need to read any further!

In some cases (e.g. if you edit the source code), it may be desirable to test some aspects of the SMOG software. In that case, you may run the three sets of checks separately with the following commands:

```
>./smog-check
```

```
>./smog-tool-check
```

```
>./scm-check
```

It may also be desirable to only employ specific checks. If you only want to test some of the tools, you need to give the names of the tools as arguments for `smog-tool-check`. For example, if you only want to test `smog_adjustPDB`, then use the command:

```
>./smog-tool-check adjustPDB
```

Similarly, you may want to run/rerun some smog-check tests. You can run a single test with the command

```
>./smog-check N
```

where N is the index of the check. You can also run a range of tests with

```
>./smog-check N M
```

Chapter 3

Using SMOG 2

This chapter describes the usage of SMOG 2. It is recommended that all users read this chapter before using the software.

3.1 Preparing the input PDB file

3.1.1 PDB file format

To prepare a SMOG force field, a structural model (e.g. crystallographic, NMR, or cryo-EM model) must be provided as a PDB file, in accordance with the PDB Content Guide. An exception is that that the coordinates may be supplied in free-format, as long as you issue the `-freecoor` flag when calling smog2.

To avoid I/O issues, please follow these additional guidelines when preparing your PDB file for use with SMOG 2:

- Only use a text editor (e.g. vi, or emacs) to prevent insertion of hidden characters.
- Earlier versions of SMOG 2 have very strict format requirements. Now, the PDB preprocessor (see next section) has been significantly enhanced, such that minimal manual editing of the PDB file is required.
- Chain identifiers are almost completely ignored. That is, if there is a change in chain ID, and there is not a TER line between the residues, then the residues will be interpreted as being in the same chain. However, all atoms in a single residue must have the same chain ID.

- Only residues and atoms within a residue defined in the force field templates will be recognized by SMOG 2. Unless a coarse-grained template is designated with -tCG, unrecognized residues and atoms will lead to a PDB parse error, and the program will exit.
- Residue name field is officially columns 18-20 in the PDB definition, but smog2 reads columns 18-21 since column 21 is unused.

3.1.2 Preprocessing the PDB file

As discussed in Chapter 6, SMOG 2 reads "template" files in order to generate force field files. As such, each PDB file has to fully conform to the molecular structure definitions provided by the templates. For example, the default all-atom templates (provided in SBM_AA) distinguish between terminal and non-terminal residues (i.e. in proteins there is an OXT in place of a peptide bond for terminal residues). In the default templates, the C-terminal amino acid residues have the suffix "T" added to their their three-letter code (*e.g.* GLY vs. GLYT).

To make preprocessing more straightforward, the `smog_adjustPDB` tool is provided. It is used to check the overall format of the PDB file and check that every residue is defined in a manner that is consistent with the force field that you plan to use for your simulation. For usage guidelines, see Section 5.1.

3.2 Generating a Structure-Based Model

SMOG 2 supports a broad range of structure-based models. The all-atom [2] and C_α [1] models are provided as defaults. See Appendix A for full details of the default models. By running SMOG, you will generate the .top, .gro, and .ndx files necessary to perform a structure-based simulation in Gromacs. These files may also be used, with limited support, in NAMD, or OpenMM. The files may also be converted to a format for use with LAMMPS. For full support in OpenMM, you must use the OpenSMOG libraries and give SMOG 2 the flag `-OpenSMOG`, which will generate an additional XML file. Additional output files are provided, for your information.

In order to use SMOG 2, the only required file is molecular structure in PDB format (typically, after preprocessing). As described below, one can then use the default models, or one may specify an alternate model by providing "template files" that define a general force field.

3.2.1 Interactive force field selection

One may specify a force field on the command line, or the force field may be selected interactively. For example, if you only provide a PDB file with the command
`> smog2 -i yourFile.pdb`
then you will be prompted to select the force field you would like to use. The `smog_ions` tool can also automatically bring up an interactive prompt.

Configuring your own custom SMOG 2 Force Field Library When using the interactive force field selection option, the force field is selected from a library. By default, the library is set to the directory `$SMOG_PATH/share/templates`. You can add more force fields to this library, or you create your own custom library.

To set up a library, you must have a directory that will serve as your library directory. Within that directory, you should have a separate directory for each set of templates that you would like to have available. Each template directory should have the following files:

- .sif file
- .bif file
- .nb file
- .b file
- .citation file (optional)
- .ions.def file (only needed for `smog_ions`)
- .map file (only needed for `smog_adjustPDB`)

In the main library directory, you must also have a file called "ff.info" that lists the template directories and provides descriptions of each force field. For an example, check the directory `$SMOG_PATH/share/templates`. If you do not want to use the default library path, you can specify a separate directory to serve as your template library by setting the shell variable `SMOG_FFDIR`.

3.2.2 Default All-Atom Model

The all-atom potential energy function is defined through the template files found in `$SMOG_PATH/SBM_AA`. These files define:

1) the covalent geometry of amino acids, nucleic acids, some ligands, as well as bioinorganic atoms
2) the energetic and system parameters (e.g. mass, charge, interaction strengths)

To generate all-atom force field and coordinate files for the default model (i.e. .top and .gro files), issue the command:

```
> smog2 -i yourFile.pdb -AA
```

where *yourFile.pdb* is the name of the file containing your molecular system.

If you would like to specify a different all-atom model, then use the command:

```
> smog2 -i yourFile.pdb -t templateDirName
```

where `templateDirName` is the name of the directory containing the desired template files.

3.2.3 Default C_α model

To generate force field and coordinate files for the default C_α model, issue the command:

```
> smog2 -i yourFile.pdb -CA
```

If you would like to use a different set of CG templates:

```
> smog2 -i yourFile.pdb -t templateDirName -tCG CGtemplateDirName
```

Note that an additional set of templates are required when using a coarse-grained model. The option `-tCG` is used to indicate the precise coarse-grained model that should be prepared. When `-tCG` is given, the `-t` flag is used to designate the templates that initially process the PDB for contact analysis. Normally an all-atom PDB is provided, since native contact maps make the most sense when generated from an all-atom structure (note that the "Shadow" map *only* makes sense with atomic graining). The `-tCG` templates are then used to construct the CG energetic model. In the above example, the PDB has residues and atoms defined in the `-t` templates, and these definition will also be used for contact map generation. See Chapter 5.8 for a detailed description of the supported contact map calculations.

3.2.4 Default Gaussian contact potential models

The default Gaussian models have energetic and structural parameters that are tailored to be similar to the LJ contact potential models, while allowing for a more flexible Gaussian contact shape. To generate force field and coordinate files for the default Gaussian all-atom model, issue the command:

```
> smog2 -i yourFile.pdb -AAgaussian
```

or for the default Gaussian C_α model,

```
> smog2 -i yourFile.pdb -CAgaussian.
```

3.2.5 Using non-default models

One of the key features of SMOG 2 is that the force fields are defined by a set of general "template" files (see Sec. 6.1). The template files must be in a single directory. The template directory may then be given to SMOG 2 with the command:

```
> smog2 -i yourFile.pdb -t templateDirName
```

where `templateDirName` is the name of the directory containing the desired template files.

3.2.6 User-provided contact map

If you have generated contacts yourself, these can be used instead of using the internal SMOG 2 routines. A single file containing all the contacts in a list can be specified at the command line with the flag `-c`. For example:

```
> smog2 -i <pdbfile> -c contacts.txt ...
```

will read the list of contacts in file `contacts.txt`

```
chainNum_i1 atomNum_i1 chainNum_j1 atomNum_j1 (opt.  distance)
chainNum_i2 atomNum_i2 chainNum_j2 atomNum_j2 (opt.  distance)
chainNum_i3 atomNum_i3 chainNum_j3 atomNum_j3 (opt.  distance)
etc ...
```

which should be formatted as a single line per contact, whitespace delimited, where each line has the two atoms that are interacting and their respective chain numbers. The chains are numbered starting from 1 by the order of occurence in the PDB file. The atomNum should be consistent with atom numbers in the input PDB file. The fifth column can contain a numeric distance in Å, which, if provided, will be used instead of the native distance. If using `-tCG` to obtain a coarse grained topology, the input contact map should designate residue numbers instead of atom numbers, again with the same numbering as in the input PDB file.

3.3 Input options

SMOG 2 always requires a PDB file. If no model is specified, then the user will have to interactively select from the available force fields. Table 3.1 shows the currently-supported input arguments.

Input Option	**Usage**	**Default value**
	Required	
`-i <string>`	input PDB file to define the model	molecule.pdb
	Optional	
`-t <string>`	folder containing templates	none
`-AA`	use the default all-atom model	N/A
`-CA`	use the default C_α model	N/A
`-AAgaussian`	use default all-atom model with gaussian contacts	N/A
`-CAgaussian`	use default C_α model with gaussian contacts	N/A
`-tCG <Folder Name>`	folder containing templates used for coarse graining.	none
`-c <string>`	input contact file name	none
`-g <string>`	output .gro file name	smog.gro
`-freecoor`	read input PDB assuming space-delimited free-formatting for coordinates	none
`-o <string>`	output .top file name	smog.top
`-s <string>`	output .contacts file name	smog.contacts
`-n <string>`	output .ndx file name	smog.ndx
`-OpenSMOG`	produce output files that are compatible with the OpenSMOG module for OpenMM	off
`-OpenSMOGxml`	output file name for OpenSMOG xml file	smog.xml
`-dname <string>`	default name to use for all output files	smog
`-backup [yes|no]`	enable/disable generation of backed up outputs	yes
`-warn [N]`	convert the first N fatal errors to warnings. Convert all errors if N=-1 (Should be used with extreme caution)	0
`-limitbondlength`	if bond length exceeds limit (in nm), set it to the limiting value	N/A
`-limitcontactlength`	if contact length exceeds limit (in nm), set it to the limiting value	N/A
`-deleteshortcontact`	if a contact length is too short, don't include it in the model	N/A
`-ignH`	ignore any atoms with name starting with 'H' in the internal contact algorithm	N/A
`-nocheck`	turn off template cross-validation checks	N/A
`-gen_map`	read the .bif file, generate a mapping file for smog_adjustPDB and exit	N/A
`-help`	show supported options	

TABLE 3.1: Flags supported by SMOG v2.6-beta

Chapter 4

Performing a simulation, or calculation

Once you have generated the .top and .gro (and possibly .xml) files with SMOG, you are ready to perform a simulation. Rather than write a new molecular dynamics simulation package, SMOG generates input files for use with Gromacs [9], NAMD [10], OpenMM and LAMMPS, highly-optimized and parallelized MD software suites. This allows you to use nearly every protocol that has been implemented in these programs when performing simulations with structure-based models (e.g. replica exchange, umbrella sampling). In addition, these packages are scalable to many processors through a combination of MPI and thread-based parallelization, and they also support GPU acceleration, which allows SMOG models to fully take advantage of cutting-edge computing resources. Here, we provide brief descriptions of how to perform SMOG model simulations with OpenSMOG/OpenMM and Gromacs, while additional external references to NAMD, OpenMM and LAMMPS are noted.

4.1 Using OpenMM

There are two different ways in which one may perform simulations with SMOG models in OpenMM.

4.1.1 Native OpenMM support for SMOG models

Some variants of the SMOG model are supported directly by OpenMM. For example, if you are using the default all-atom models, then (for many systems) you can run a simulation by following the example provided on the Grossfield Group webpage.

4.1.2 OpenSMOG module for OpenMM

Starting with version 2.4, SMOG 2 has the OpenSMOG option implemented (Described in [11]), which allows users to generate force fields for use with OpenMM. This extends far beyond native-OpenMM support and allows essentially all variants of SMOG models to be used. When using this flag, SMOG 2 will generate input decks that are written for use with the OpenSMOG module for OpenMM, which is available through conda, pip, or source-code installation. Information on OpenSMOG is available at smog-server.org. Below, we describe the OpenSMOG framework and how it can be used to perform simulations with existing SMOG models, as well as how to use SMOG-model variants in OpenMM.

4.1.2.1 Basic usage

Using OpenSMOG involves two primary steps. First, one must tell SMOG 2 to provide force field files that are properly formatted for use with the OpenSMOG module of OpenMM. This is achieved by using the flag -OpenSMOG with SMOG 2. This flag is compatible with all templates that are distributed with SMOG 2. Since the OpenSMOG flag does not change the force field, the exact same models may be used in Gromacs (when supported), or OpenMM . The only difference when using -OpenSMOG is that one additional XML-formatted file is produced by SMOG 2. While Gromacs only requires a gro and top file, OpenSMOG uses the gro, top and an XML file to define a model. Once these files have been generated, they may be used directly with the OpenSMOG module of OpenMM. For a step-by-step tutorial that describes how to use the OpenSMOG module in OpenMM, see the OpenSMOG documentation or OpenSMOG Tutorials page.

4.1.2.2 Advanced usage (User-defined Custom Potentials)

The power of the OpenSMOG framework is that SMOG 2 has been redesigned to interface with the CustomForce framework that is supported by OpenMM. With this extension to SMOG 2, one may define customized potential energy functions in SMOG 2 and directly use them in OpenMM. When the -OpenSMOG flag is given to SMOG 2, your force field definitions will be applied to your molecular system and then formatted for use with the OpenSMOG module of OpenMM. In addition to the description below, there are numerous OpenSMOG tutorials available at the OpenSMOG Tutorials page.

Beginning with SMOG v2.4, users can define their own functional form for pairwise native contact interactions. Support for custom nonbonded interactions (i.e. non-specific non-contact interactions) and global constants was added in SMOG v2.4.2. Custom

dihedrals were added in v2.5, while custom bond angles and external potentials were added in v2.6. We plan to include support for multi-body interactions in future versions. Here, we describe examples for how to introduce a range of different potentials for contact, non-bonded, bond angles and dihedrals in OpenSMOG.

N-M contact potentials: While the all-atom model uses a 6-12 potential for native contacts and the C_α model uses a 10-12 potential, these only represent two possible potentials that one may want to employ. If you would like to use an arbitrary N-M potential (eq. 6.1), then you only need to make a few changes in the template files. For example, if you want to use an 8-14 potential, one only needs to change a single line in the .nb file of the templates. In the default all-atom model, the contact potential is defined with the following line in the .nb file:

```
1  <contact func="contact_1(6,12,?,energynorm)" contactGroup="c">
2    <pairType>*</pairType>
3    <pairType>*</pairType>
4  </contact>
```

LISTING 4.1: Defining the contact potential in the default all-atom model: AA-whitford09.nb

The contact potential is defined by the `contact_1` function, which is defined to use the exponents 6 and 12. The question marks indicate that native contact distances are used to define the position of the minimum, `energynorm` indicates that the weights should be normalized. If one wanted to change the model to use an 8-14 potential, these lines would simply need to read:

```
1  <contact func="contact_1(8,14,?,energynorm)" contactGroup="c">
2    <pairType>*</pairType>
3    <pairType>*</pairType>
4  </contact>
```

LISTING 4.2: Redefining the contact potential in the default all-atom model

In this definition, contact_1 assigned coefficients such that the position is at the native distance, and the depth of the well is unaltered from the original definition.

User-defined contact potentials: In addition to supporting N-M potentials, Gaussian potentials and harmonic potentials for contacts, the OpenSMOG framework allows the user to define nearly any pairwise function to be used with native contacts. For example, perhaps one would like to use a potential given by:

$$A\left(\frac{1}{r^{16}} + \tanh(B(r-C))\right) \tag{4.1}$$

where A, B and C are parameters for each contact. To implement this potential, you would first need to define the function in the .sif file of your templates. For this particular

function, one would add the following child element to the functions element in the .sif file:

```
<function name="exp_tanh"
  directive="OpenSMOG"
  OpenSMOGtype="contact"
  OpenSMOGpotential="weight*(1/r^16+tanh(B*(r-sigma)))"
  OpenSMOGparameters="weight,B,sigma"
  exclusions="1"
/>
```

LISTING 4.3: Defining a new contact potential

In this example, we used the term "weight" for the prefactor in the potential. This is not always necessary, but it is if contact strengths are to be normalized by smog2. The directive "OpenSMOG" indicates that this is a function that is only to be used with the OpenSMOG module of OpenMM. For custom contacts, OpenSMOGtype must be set to "contact". The value given with OpenSMOGpotential is the functional form of the interaction. This definition must adhere to the conventions supported by CustomBondForce in OpenMM, since the expressions will be passed directly to these routines. Similar to how SMOG models are applied in Gromacs, OpenSMOG uses "bonded" routines to define native contacts. However, these interactions are non-bonded, in character. They are simply labeled as bonded. OpenSMOGparameters tells smog2 which terms need to be given values for each interaction. The order of the parameters is important, since it defines how one needs to invoke this function. The name is simply a string used for referring to the function. Finally, "exclusions" indicates whether atom pairs associated with contacts of this type should be excluded from non-bonded calculations. This means, should the model include the specific contact, in addition to the non-specific non-bonded (i.e. derived by atom types, generally) interactions. Typically, one does not want these additional non-specific terms, which requires that exclusions are added (i.e. exclusions=1). If you don't want to generate exclusions, then set "exclusions" to 0 in the .sif file. In SMOG v2.5, and earlier versions, SMOG2 would list exclusions under the exclusions directive of the top file. Starting with v2.6, exclusions are generated at OpenSMOG runtime, if the exclusions tag is present in the OpenSMOG input file (generated by SMOG 2).

After the function has been defined (e.g. exp_tanh), you may use it in the .nb file, where specific types of pair types will be defined to interact through this potential. For example, this function could be called

```
<contact func="exp_tanh(energynorm,1,?)" contactGroup="c10">
  <pairType>P_10</pairType>
  <pairType>P_10</pairType>
</contact>
```

LISTING 4.4: Using a new contact potential

In this case, any pairs of atoms with pairType of P_10 will interact using this potential. Since the parameters were listed in the order weight, B, sigma, the "energynorm,1,?" indicates that normalized weights and native distances (?) should be used. Normalization of weights is explicitly turned on with the following line in the .sif file:

```
1       <contactGroup name="c10" intraRelativeStrength="1" normalize="1"/>
```

LISTING 4.5: turning on normalization for a custom contact potential

User-defined dihedral potentials: In addition to cosine functions and harmonic potentials, the OpenSMOG framework allows the user to define nearly any function to be used with dihedral potentials. For example, perhaps one would like to use a potential given by:

$$A\left(1-\cos^2(B*(\phi-C))\right) \tag{4.2}$$

where A, B and C are parameters for each dihedral. To implement this potential, you would first need to define the function in the .sif file of your templates. For this particular function, one would add the following child element to the functions element in the .sif file:

```
1     <function name="cossquared"
2       directive="OpenSMOG"
3       OpenSMOGtype="dihedral"
4       OpenSMOGpotential="weight*(1-(cos(multiplicity*(theta-theta0))^2))"
5       OpenSMOGparameters="theta0,weight,multiplicity"
6     />
```

LISTING 4.6: Defining a new dihedral potential

When using custom potentials, one must given functional forms that adhere to the conventions supported by CustomTorsionForce. For example θ is the dihedral angle, in radians. We have tried to allow for all OpenMM-supported functions to be allowed when using OpenSMOG. Once the function is defined in the .sif file, it may be used to assign terms to dihedrals in the .b file.

User-defined non-bonded (i.e. non-contact) potentials: In SMOG v2.4.2 (OpenSMOG v1.1), we added support for custom non-bonded potentials. To introduce a custom non-bonded potential, one needs to define the `CustomNonBonded` element in the .sif file. As an example, the following listing shows how one would define all non-bonded interactions to be composed of an $\frac{1}{r^{12}}$ term, Coulomb electrostatics and a single Gaussian potential.

```
<CustomNonBonded OpenSMOGparameters="C12,B1,C1,R1"
      OpenSMOGcombrule="none"
      OpenSMOGpotential="K_coul*q1*q2/dielectric*(1/r-1/r_c)
                        +C12*(1/r^12-1/r_c^12)
                        +B1*exp(-C1*(r-R1)^2)"
                        r_c="2.0"/>
```

LISTING 4.7: Using a new nonbonded potential. OpenSMOGcombrule is required, but it currently can only be set to "none". r_c is optional, but it allows one to require OpenMM to use a specified nonbonded cutoff for any models generated with these templates. That is, if r_c is set, then the cutoff value will be passed to OpenMM automatically.

The functional form must adhere the standards of CustomNonbondedForce in OpenMM.

To pass values for the constants, add the `OpenSMOGsettings` element in .sif file.

```
<OpenSMOGsettings>
  <constants>
    <!-- Note: In this example, K_coul is 1/5 of the value in most software
               since SMOG models use reduced units, where room temperature
               corresponds to 0.5.
    -->
    <constant name="K_coul" value="27.787097"/>
    <constant name="dielectric" value="80"/>
  </constants>
</OpenSMOGsettings>
```

LISTING 4.8: Defining constants for use in OpenSMOG potentials

As a note, the constants may also be used in any OpenSMOG potential function (i.e. contacts, or non-bonded terms). Also, the symbol r_c is reserved by OpenMM to indicate the cutoff distance, which may be given in the nonbonded function definition.

To specify the parameters for individual interactions, include the relevant lines in the nonbond_param entries of the extras file. Each line must provide two atom types, a function type index (doesn't affect OpenSMOG) and then values of the parameters (in the order defined by `OpenSMOGparameters`).

As noted above, one may optionally use the templates to define the cutoff distance for the model. Specifically, add the `r_c` attribute to the `CustomNonBonded` element, where the value is the cutoff distance (nm). This will ensure that, if the model is used in OpenSMOG, then the proper cutoff will be applied.

4.2 Using Gromacs 4.5 or 4.6

4.2.1 All-Atom Model

First, produce a portable xdr file (in the example below, run.tpr) that describes your simulation. This file is platform-independent and contains all parameters for your simulations. This allows you to produce a tpr file on any machine, and then move it to another machine and run your simulation. The xdr file is produced by grompp (part of the Gromacs distribution):

```
> grompp -f mdpfile.mdp -c gro_file.gro -p top_file.top -o run.tpr
```

The file `mdpfile.mdp` tells Gromacs what settings to use during the simulation, such as the timestep size, the number of timesteps and what thermostat to use. Here is a sample set of recommended configurations when using the default all-atom model:

```
integrator = sd ;Run control: Use Langevin Dynamics protocols.
dt = 0.002 ;time step in reduced units.
nsteps = 100000 ;number of integration steps
nstxout = 100000 ;frequency to write coordinates to output trajectory .trr file.
nstvout = 100000 ;frequency to write velocities to output trajectory .trr file
nstlog = 1000 ;frequency to write energies to log file
nstenergy = 1000 ;frequency to write energies to energy file
nstxtcout = 1000 ;frequency to write coordinates to .xtc trajectory
xtc_grps = system ;group(s) to write to .xtc trajectory (assuming no ndx file is supplied to grompp).
energygrps = system ;group(s) to write to energy file
nstlist = 20 ;Frequency to update the neighbor list
coulombtype = Cut-off
ns_type = grid ; use grid-based neighbor searching
rlist = 1.2 ;cut-off distance for the short-range neighbor list
rcoulomb = 1.2 ; cut-off distance for coulomb interactions
rvdw = 1.2 ; cut-off distance for Vdw interactions
pbc = no ; Periodic boundary conditions in all the directions
table-extension = 10 ; (nm) Should equals half of the box's longest diagonal.
tc-grps = system ;Temperature coupling
tau_t = 1.0 ; Temperature coupling time constant. Smaller values = stronger coupling.
ref_t = 60.0 ; ~1 reduced temperature unit (see Gromacs manual or SMOG 2 manual for details)
Pcoupl = no ;Pressure coupling
gen_vel = yes ;Velocity generation
gen_temp = 60.0
gen_seed = -1
ld_seed = -1
comm_mode = angular ; center of mass velocity removal.
```

LISTING 4.9: Sample mdp file for all-atom SMOG models used for Gromacs v4.5/4.6

Note: If you would like to perform energy minimization, simply change the `integrator` settings to `steep` (steepest descent) or `cg` (conjugate gradient).

After you have generated the .tpr file with `grompp`, you will need to perform the simulation. To run the simulation, issue the command:

```
> mdrun -s run.tpr -noddcheck
```

Is is highly recommended that you explore all Gromacs options, in order to ensure maximum performance (e.g. the number of threads being used). **SMOG-model specific requirement:** To use domain decomposition when performing a simulation in parallel, using either threads, or MPI, you should add the additional flag `-noddcheck`. Note: For folding of small proteins you will probably want to avoid domain decomposition, and instead use particle decomposition by adding the option `-pd` when on a single node.

4.2.2 C_α Model

To run a simulation with the C_α model, the steps are the same as for the AA model, though there are a few minor differences. First, when running `grompp`, you will want to change a few settings in the .mdp file. A sample .mdp file for C_α models is given below.

```
integrator = sd ;Run control: Use Langevin Dynamics protocols.
dt = 0.0005 ;time step in reduced units.
nsteps = 100000 ;number of integration steps
nstxout = 100000 ;frequency to write coordinates to output trajectory .trr file.
nstvout = 100000 ;frequency to write velocities to output trajectory .trr file
nstlog = 1000 ;frequency to write energies to log file
nstenergy = 1000 ;frequency to write energies to energy file
nstxtcout = 1000 ;frequency to write coordinates to .xtc trajectory
xtc_grps = system ;group(s) to write to .xtc trajectory (assuming no ndx file is supplied to grompp).
energygrps = system ;group(s) to write to energy file
nstlist = 20 ;Frequency to update the neighbor list
coulombtype = Cut-off
ns_type = grid ; use grid-based neighbor searching
rlist = 3.0 ;cut-off distance for the short-range neighbor list
rcoulomb = 3.0 ; cut-off distance for coulomb interactions
rvdw = 3.0 ; cut-off distance for Vdw interactions
coulombtype = User
vdwtype = User
pbc = no ; Periodic boundary conditions in all the directions
table-extension = 10 ; (nm) Should equals half of the box's longest diagonal.
tc-grps = system ;Temperature coupling
tau_t = 1.0 ; Temperature coupling time constant. Smaller values = stronger coupling.
ref_t = 80.0 ; ~1 reduced temperature unit (see Gromacs manual or SMOG 2 manual for details)
Pcoupl = no ;Pressure coupling
gen_vel = yes ;Velocity generation
gen_temp = 80.0
gen_seed = -1
ld_seed = -1
comm_mode = angular ; center of mass velocity removal.
```

LISTING 4.10: Sample mdp file for C_α SMOG models used for Gromacs v4.5

The most significant difference is the use of "User-defined" vdW and Coulomb interactions. This is due to the fact that the 10-12 potential is used for contact interactions in the C_α model. In order to run `mdrun` (next step), it is necessary to generate table files that define the 10-12 interaction. We provide a tool for generating these tables (`$SMOG_PATH/bin/smog_tablegen`) with the SMOG 2 distribution, which is described in Section 5.7.

After you have generated your tabulated potentials for the 10-12 interaction (i.e. table_file.xvg) and you have prepared a .tpr file with `grompp`, you can run the simulation with the command:

```
> mdrun -s run.tpr -noddcheck -table table_file.xvg -tablep table_file.xvg
```

Typically, for protein folding, you will want to avoid domain decomposition and instead use particle decomposition by adding the option `-pd` when on a single node. After you perform your simulation, you can utilize any analysis tools provided with Gromacs.

4.2.3 Examples

Check $SMOG_PATH/examples/gromacs4 for some complete examples with terminal history.

4.2.4 Note on Domain Decomposition

The `[ pairs ]` section is treated as a bonded interaction by Gromacs and therefore all pairs within a single domain are always calculated regardless of the .mdp parameter `rvdw`. If you are using `-pd` with version 4.X or only OpenMP threads in version 5.0, you can set `rvdw` to only take into account the non-bonded excluded volume. For the default models this would be 0.6 nm for all-atom and 1.0 nm for Calpha. These lengths are derived as $\approx 2.5 r_{\mathrm{NC}}$.

4.3 Using Gromacs 5

Gromacs 5 has a few changes that impact SMOG models. First, we don't yet provide a Gromacs 5 distribution with the SMOG enhancements (umbrellas, `g_kuh`, gaussian contact potentials). So, if you want to use these you can only use Gromacs 4.5. Gromacs 5 itself has changes of note: 1) OpenMP support has replaced the option of particle decomposition and 2) OpenMP requires `cutoff-scheme=Verlet` and Verlet doesn't yet

allow tabulated potentials. This has the largest impact on C_α models, which use tabulated potentials. If your simulated system has less than roughly 100 atoms, you can typically only use a single processor with v5, because additional threads are only allowed through OpenMP. If your system is large enough, you can use multiple MPI processes with domain decomposition to scale to multiple cores. When using Verlet lists you have to use `pbc = xyz`. For all-atom simulations, Verlet lists are fine, and it is usually best to use as many OpenMP threads as possible with `-ntomp`.

4.3.1 Examples

Check $SMOG_PATH/examples/gromacs5 for complete examples, including terminal history.

4.4 Using Gromacs 2020

From an interface standpoint, nothing significant changed between 5 and 2020. While the Gromacs developers have added many new options and have introduced performance improvements, running SMOG models is largely unaltered.

4.4.1 Examples

Check $SMOG_PATH/examples/gromacs2020 for complete example, including terminal history.

4.5 Using NAMD

For some model variants, the force field files generated by SMOG 2 are fully compatible with NAMD. To perform SMOG models in NAMD, please consult the NAMD manual. For questions about running SMOG simulations in NAMD, please contact the NAMD support team.

4.6 Using LAMMPS

The force field files generated by SMOG 2 may be converted to a format that is compatible with LAMMPS through use of the program (SMOG-converter). For questions about usage, please consult the SMOG-converter developers.

4.7 Discrete Path Sampling (only available in Beta version)

An alternate method for sampling the landscape is to use Discrete Path Sampling (DPS). As part of the SMOG 2 package, a script (`smog_optim`) is provided that will convert the .gro and .top files into the inputs necessary for DPS using the OPTIM/PATHSAMPLE suite developed by the Wales Group. Currently supported interactions are: bonds, bond angles, cosine and harmonic dihedrals, 10-12 and 6-12 contacts, anisotropic position restraints and Debye-Hückel electrostatics. As we develop and test the protocols for `smog_optim`, this manual will be updated with recommended step-by-step instructions for DPS SMOG models. For now, we just provide a few examples for how to get started.

Before discussing usage, there are a few important differences between the SMOG force fields used in Gromacs, and those applied in DPS.

1. Even though the routines required for using DPS with SMOG models are invoked with the "SB" flag (see listing 4.11), these routines are not structure-based-specific. That is, these are general routines for calculating bond, angle, dihedral, contacts, excluded volume and electrostatic energies/forces. Similar to how SMOG models are implemented in Gromacs, the input force field file (SBM.INP) encodes the structure-based aspects of the model. Thus, one may define a SBM.INP file that has many non-structure-based features (e.g. non-specific contacts and electrostatics), even though the "SB" routines would be used.

2. The SB routines in OPTIM apply switching functions to non-bonded terms. The switching range (r_0 to r_c) is defined on the third line of SBM.INP. The switching function is defined as a fourth-order polynomial, which ensures the force continuously reaches zero at r_c.

3. In Gromacs, one may exclude all non-bonded interaction that are connected by less than a specific number (nrexcl) of bonds. In OPTIM, exclusions are automatically generated for all atoms that interact via a bond, bond angle, dihedral, or contact. nrexcl is not used.

To convert the .gro and .top files into an odata and SBM.INP file, which is necessary for OPTIM/PATHSAMPLE, use the `smog_optim` module (described in section **??**). This tool may be found in the `$SMOG_PATH/bin` directory. The script is interactive, and you will be prompted for all necessary input and options. Once you have organized your force field for DPS, you will need to perform energy minimization for your structure.

By default, the `odata` file generated by `smog_optim` will provide the keywords for minimization. The `odata` file contains the initial coordinates, as well as the calculation specifications. The `odata` file should look like the following:

```
1  STEPS 1000000
2  BFGSMIN 0.000001
3  POINTS
4  SB 13.67 48.35 -15.2
5  SB 12.16 51.69 -14.1
6  SB 10.07 51.44 -10.94
7  SB 6.78 53.35 -10.85
8  SB 4.22 53.53 -8.01
9  SB 0.53 53.56 -8.78
10 ...
```

LISTING 4.11: Header of an example `odata` file generated by `smog_optim`

If you selected rigidification when using `smog_optim`, then there will be the additional keyword `RIGIDINIT` in the `odata` file. In addition to the `odata` file, `smog_optim` will also generate the file `SBM.INP`, which defines the SMOG force field.

After energy minimizing two structures, you will want to find a connection between them. To find an initial connection, you need to update the `odata` file, such that it starts with:

```
1  NEWCONNECT 200 20 1.0 50.0 20 2.0 0.025
2  MAXERISE 1.0D-4 1.0DO
3  NOPOINTS
4  USEDIAG 2
5  NEWNEB 15 100 0.025
6  NEBK 500
7  DIJKSTRA
8  MAXBFGS  0.4 2.0
9  EDIFFTOL 0.000005
10 DUMPALLPATHS
11 PATH     100 0.001
12 MAXSTEP 0.05
13 MAXMAX  1.25
14 TRAD 0.2
15 BFGSTS  500 3 20 0.001
16 NOHESS
17 ENDNUMHESS
18 BFGSMIN  0.000005
19 PUSHOFF 0.1
20 STEPS    20000
21 BFGSSTEPS 1000000
22 UPDATES 200
23 POINTS
24 SB ...
```

LISTING 4.12: Example `odata` file specifying a double-ended search.

In this case, the coordinates in the `odata` file should correspond to the coordinates of one of the minimized endpoints. The coordinates of the second endpoint should be provided in a file called `finish`. `finish` should simply be a listing of XYZ coordinates (no "SB" at the beginning of each line). These steps are sufficient to get one started with DPS using models generated by SMOG 2. Note: Since SMOG models are so computationally inexpensive, compared to other models, I/O can sometimes become limiting when using OPTIM. It is advisable that you check the summary statistics at the end of each use of OPTIM and verify that roughly 90 percent of time is spent on energy+gradient calls.

Chapter 5

SMOG Tools

In addition to applying standard structure-based models, there is often a need to introduce additional system-specific modifications to the force field. To help users implement some less-than-trivial tasks, we provide a variety of additional scripts for force field and system modification. If `$SMOG_PATH/bin` is in your `PATH`, then the smog tools should already be available on your machine.

5.1 `smog_adjustPDB`

This script helps rename atoms and residues in a PDB file, such that they will conform to the naming scheme used in a set of template files. To see all options, use the `-h` flag, or refer to Table 5.1.

The only required flag is the input PDB file name. You may optionally also explicitly indicate which mapping file to use with `-map`. If you do not give the `-map` flag, then you will be prompted to select which force field you would like to use. The available force fields will be determined by your local force field library. For instructions on how to set up your own custom library, see Sec. 3.2.1.

map file format Starting with SMOG v2.3, the default behavior of `smog_adjustPDB` has changed. Previously, only terminal residues were renamed. Starting with v2.3, the following changes have been introduced:

- An exact matching protocol is now used to determine mapped residue names.
- Support was added for alternate atomic naming conventions (e.g. C3* or C3').

- One can now ignore (not rename) specific residue names, regardless of the atomic composition.

Currently, there are 4 types of lines that can appear in the mapping file:

- **comment lines**: A semicolon can be used to denote a comment. If there are only white spaces before a comment, then the line will be ignored.
- **rename**: This is used to globally rename a specific atom. For example, one may want to use the prime convention, or asterisk convention. As another example, if you wanted to rename all CA atoms as CAA, then you would give the line:

  `rename CA CAA`

  The convention is to begin with `rename` (not case-sensitive), followed by the atom name that may appear in the PDB and then the desired atom name (i.e. the name used in the force field).
- **ignoreres**: Sometimes one does not want to rename specific residues. Example: If one wanted to not rename PSU residues, nor its atom names, then the following line would be added:

  `ignoreRes PSU`
- **residue definitions**: To define the composition of a residue, start with `residue`, followed by the target residue name and then list all atoms that should appear (i.e. are defined in the associated force field). For example, one could define ALA in the following way:

  `residue ALA C CA CB N O`

  Based on this, any residue that has (only) a C, CA, CB, N and O would be renamed ALA.

  Note that the `rename` option is applied before matching the composition to a residue. So, if you had a rename definition that converted CF to CA (`rename CF CA`), then you would also match ALA if the residue has C, CF, CB, N and O. In this example, `C CA CB N O` or `C CF CB N O` would both match ALA. The output PDB file would reassign the CF atom to be CA.

  There is a second strategy to allow for multiple types of atom sets to correspond to the same target residue. For example, one may encounter a PDB file that uses O2 to refer to OXT. SMOG 2 could enable both atom names in a single residue definition via:

  `residue ALAT C CA CB N O (OXT O2)`

This would recognize `C CA CB N O OXT` or `C CA CB N O O2` as ALAT. Since `OXT` is listed before `O2`, the output PDB file would call the atom `OXT`.

A final option with the residue definition line is the use of `%first` or `%last`. These tags indicate that, in addition to matching the atoms, the corresponding residue name will only be substituted if the residue is the first/last residue in a chain. This may occur if, for example, one is assigning charges to RNA residues. Since the terminal residues may have the same atoms, but different charges, then one may need to assign a different name for the terminal position. Note: If the chain is a single residue, any definition with `%first` or `%last` is not considered when matching the atoms.

To see the current default format, check out the file `share/mapfiles/sbmMapExact`.

```
1  ;This is a mapping file that was generated by smog2
2  ;This defines the composition of atoms in the templates found in:
3  ;   /Users/coolsim/smogtemplates
4  rename OP1 O1P
5  rename O3' O3*
6  residue A C1* C2 C2* C3* C4 C4* C5 C5* C6 C8 N1 N3 N6 N7 N9 O1P O2* O2P O3* O4* O5* P
7  residue ALAT C CA CB N (O O1) (OXT O2)
8  residue A C1* C2 C2* C3* C4 C4* C5 C5* C6 C8 N1 N3 N6 N7 N9 O1P O2* O2P O3* O4* O5* P
9  residue A3P C1* C2 C2* C3* C4 C4* C5 C5* C6 C8 N1 N3 N6 N7 N9 O1P O2* O2P O3* O4* O5* P \%last
10 ....
```

LISTING 5.1: example mapping file format

Generating the mapping file: While one can make the mapping file by hand, SMOG 2 can also generate a mapping file that defines all of the residues in a given force field. To generate a minimal mapping file for a given .bif file, use the `gen_map` option with `smog_adjustPDB`. Note: This will only generate the `residue` definitions, listing the exact atom names defined in the force field (i.e. `rename`, `ignoreres` and `()` will not generated). If a residue has a "meta" attribute defined in the .bif file, then its value will be included at the end of the atom names with a "%". For example, if you have a "last" definition, then `meta=''last''` in the residue definition in the .bif file would lead to "%last" being included in the residue definition of the SMOG-generated mapping file.

Legacy map file format (not recommended) When using the `-legacy` option, the following format should be used for the map file: Lines containing a "#" character are interpreted as comments. Each line must have three strings that are space/tab delimited. The first field is the residue name, as it appears in the input PDB file. The second is the name to be substituted if the residue is the first residue in a chain (e.g. N-terminus in a protein), and the third field is the corresponding substitution for the last residue (e.g. C-terminus) in each chain. The preprocessing tool will write a modified PDB file `smog.pdb`.

Option	Usage	Default value
Required		
`-i <string>`	input PDB file name	none
Optional		
`-map <string>`	user-defined mapping file name	N/A
`-legacy`	use non-matching routines	N/A
`-removeH`	strip PDB of any atoms beginning with "H"	N/A
`-noautorename`	do not automatically replace * and ' characters in atom names	N/A
`-removewater`	strip PDB of any residues named "HOH" or "WAT"	N/A
`-altLocs`	If alternate locators are found, only keep the first entry for each atom	N/A
`-insertTER`	interactively insert TER lines between non-consecutive residue numbers	N/A
`-large`	use base-N (N>10) numbering for residues and atoms	N/A
`-sort`	reorder atoms in each residue alphabetically by name	N/A
`-PDBresnum`	keep original PDB residue numbering	N/A
`-renumber`	automatically renumber all non-consecutive residues	N/A
`-subALA`	rename a residue as ALA if it contains only C, CA, N, C and CB	N/A
`-gen_map <string>`	read a template and generate the corresponding mapping file (rarely needed). output name is arg.	N/A
`-t <string>`	templates to read, when using gen_map flag	N/A
`-o <string>`	output PDB file name	adjusted.pdb
`-warn [N]`	convert N errors to warnings	0
`-help`	show options	

TABLE 5.1: Flags supported by `smog_adjustPDB`

The script can also renumber atom and residue indices to be sequential within each chain, and it adjusts atom names to be consistent with the `SBM_AA` template files.

5.2 `smog_editgro`

This tool is only useful if you are using OpenSMOG. If you are using Gromacs, the same functionality is available through the editconf tool. `smog_editgro` is used to customize

Option	Usage	Default value
`-g <string>`	input .gro file	smog.gro
`-og <string>`	output .gro file	smog.box.gro
`-boxtype <string>`	Type of box to create: cubic, rectangular, dodecahedron, octahedron	N/A
`-d`	minimum distance between system and box edges	0
`-c`	center the system in the box	
`-pbc`	impose periodic boundary conditions on the coordinates. Note: if -c is used, then PBCs are applied after centering.	
`-help`	show options	

TABLE 5.2: Flags supported by `smog_editgro`

the simulated box and initial coordinates. To see all options, use the `-h` flag, or refer to Table 5.2.

5.3 `smog_extract`

It is common when studying large molecular assemblies that you will only want to simulate a portion of the system. In these cases, it is often convenient to remove many atoms from the model, and then apply position restraints on the boundary atoms [12]. To facilitate this, `smog_extract` will take a SMOG .top and .gro file (.xml file, if using OpenSMOG), and produce a new set of force field files that only include a specified subset of atoms. The atom list can be given in a Gromacs-style .ndx file. If multiple groups are listed in the ndx file, the user will be prompted to select a single group. By default, position restraints are not introduced. If you would like to include position restraints on all atoms that have an interaction removed during extraction, then use the `-restraints` flag to indicate the strength of the restraints. To see all options, use the `-h` flag, or refer to Table 5.3.

5.4 `smog_ions`

This module allows one to add ions to your model. There are multiple ways in which the definitions used for the ions can be indicated by the user. In all approaches, vdW parameters, mass, charge and names must all be specified. The ions will be place around the existing atoms without introducing atomic clashes.

Option	Usage	Default value
`-f <string>`	input .top file	smog.top
`-g <string>`	input .gro file	smog.gro
`-n <string>`	index file for group definitions	smog.ndx
`-of <string>`	output .top file	extracted.top
`-og <string>`	output .gro file	extracted.gro
`-openSMOG <xml file>`	process an openSMOG xml file generated by smog2	none
`-openSMOGout <xml file>`	output file name for processed openSMOG xml file	none
`-om <string>`	mapping between orig. and new system	atomindex.map
`-orm <string>`	mapping between orig. and new system for restrained atoms	restrained.map
`-restraints <float>`	turns on restraints on boundary atoms	N/A
`-ndxorder`	preserve atom ordering given in the ndx file, rather than in the top.	N/A
`-warn [N]`	convert first N errors to warnings	0
`-nogro`	only convert a top file	N/A
`-help`	show options	

TABLE 5.3: Flags supported by `smog_extract`

There are three ways to specify the parameters of ions that one wants to add:

- Specify the force field that you want to use with the -t option. You must supply a template directory. Within the directory, you must have an "ions.def" file (see Sec. 6.2.6). The template directory may also contain an extras file (see Sec. 6.2.5).
- Specify the parameters of the new ions on the command line (not recommended).
- If you do not specify the parameters or templates, then you will be prompted to interactively select a force field. Note that this functionality only works if you have set up a local SMOG 2 force field library with force fields that define ions. For information on force field libraries, see Sec. 3.2.1.

For a full list of supported flags, see Table 5.4.

5.5 `smog_modifyXML`

This tool uses an OpenSMOG XML file, along with an index file to generate a new OpenSMOG XML file in which contact and/or dihedral parameters are modified. The

Option	Usage	Default value
`-f <string>`	input .top file	smog.top
`-g <string>`	input .gro file	smog.gro
`-OpenSMOG <string>`	input .xml	N/A
`-of <string>`	output .top file	smog.ions.top
`-og <string>`	output .gro file	smog.ions.gro
`-OpenSMOGout <string>`	output .xml	N/A
`-ionnm <string>`	name of the ion to be added	
`-ionn <integer>`	number of ions to add	
`-ionq <float>`	charge of ions	
`-ionm <float>`	mass of ions	none
`-ionC12 <float>`	non-bonded C12 parameter for ions	
`-ionC6 <float>`	non-bonded C6 parameter for ions	0.0
`-mindist <float>`	minimum distance an ion may be placed to any other atom	0.5 nm
`-t <string>`	template directory for reading ion parameters	none
`-warn [N]`	convert first N errors to warnings	0
`-help`	show options	

TABLE 5.4: Flags supported by `smog_ions`

script can be used interactively, or through command-line arguments. One may apply an arbitrary number of modifications to an arbitrary number/set of parameters. The specific interactions that will be rescaled are determined by the index file and user input at runtime. If you specify that dihedrals will be modified, you will be prompted to select an index group for rescaling. If the atoms forming the middle bond are within the index group, then the indicated parameters will be modified. If one is modifying contacts, then the user will need to select two index groups. Any contacts between the two groups will be modified. If the same group is selected twice, then intra-group contacts will be modified. Table 5.5 describes the available options.

5.6 `smog_scale-energies`

This tool uses a SMOG .top and .gro file, along with an index file to generate a SMOG model in which contacts and/or dihedrals weights are modified. This is a common task when using SMOG models, and we provide this script as a convenience. On the command line, you must indicate whether you want to rescale dihedrals (`-rd <float>`) or contacts (`-rc <float>`). A value of 0 indicates the interaction should be deleted from the model. The specific interactions that will be rescaled are determined by the index file and user input at runtime. That is, if you specify that dihedrals will be rescaled,

Option	Usage	Default value
`-OpenSMOG <string>`	input .xml file	OpenSMOG.xml
`-OpenSMOGout <string>`	output .xml file	OpenSMOG.out.xml
`-n <string>`	index file	smog.ndx
`-inter <string>`	type of interactions to modify (contacts, or dihedrals)	
`-type <string>`	which set of interactions should be modified (as named in XML file)	
`-grp1 <string>`	name of first group of atoms (as named in .ndx file)	
`-grp2 <string>`	name of second group of atoms (only for use with contacts)	
`-param <string>`	which parameter should be modified	
`-modby <string>`	expression for modifying parameter. Format: `[+-*/]<number>`	
`-remove`	remove the matching interactions, instead of adjusting them	
`-warn <int>`	convert first N errors to warnings	0
`-help`	show options	

TABLE 5.5: Flags supported by `smog_modifyXML`

Option	Usage	Default value
`-f <string>`	input .top file	smog.top
`-n <string>`	index file	smog.ndx
`-of <string>`	output .top file	smog.rescaled.top
`-rc <float>`	rescale contact weights by factor	1.0
`-rd <float>`	rescale dihedrals weights by factor	1.0
`-help`	show options	

TABLE 5.6: Flags supported by `smog_scale-energies`

you will be prompted to select an index group for rescaling. If all four atoms that form a dihedral are within the index group, then the weight is rescaled. Only dihedrals of type 1 are rescaled. If one is rescaling contacts, then the users will need to select two index groups. Any contacts between the two groups will be rescaled. If the same group is selected twice, then intra-group contacts will be rescaled. Table 5.6 describes the available options.

Option	Usage	Default value
	Optional	
`-N <integer>`	exponent of attractive non-bonded interaction	6
`-M <integer>`	exponent of repulsive non-bonded interaction	12
`-ic <float>`	total monovalent ion concentration (Molar) for DH interaction	0
`-temp <float>`	simulation temperature corresponding to room temperature (Gromacs units)	300
`-units <float>`	units to be used in the simulation (kCal or kJ)	kCal
`-sd <float>`	distance (nm) to start switching function for electrostatics	1.0
`-sc <float>`	distance (nm) at which switching function enforces elec. interactions go to zero	1.5
`-tl <float>`	length (nm) of table	5
`-table <string>`	output table file name	table.xvg
`-help`	show options	N/A

TABLE 5.7: Flags supported by `smog_tablegen`

5.7 `smog_tablegen`

This is only to be used if you are running simulations in Gromacs. If you are using OpenSMOG/OpenMM, then this tool cannot be used. When using user-defined potentials (i.e. not 6-12, or direct Coulomb interactions), then it is necessary to provide a table file that contains tabulated potentials and forces. Specifically, Gromacs will consider any tabulated potential of the form:

$$U = q_i q_j f(r_{ij}) - Bg(r_{ij}) + Ah(r_{ij}) \tag{5.1}$$

where A and B are defined in the .top file, and q_i is the charge of atom i. The functions $f(r_{ij})$, $f'(r_{ij})$, $g(r_{ij})$, $g'(r_{ij})$, $h(r_{ij})$, $h'(r_{ij})$ are given by the table file.

`smog_tablegen` will generate a table file with specific parameters, where $h(r_{ij}) = \frac{1}{r_{ij}^M}$, $g(r_{ij}) = \frac{1}{r_{ij}^N}$ and $f(r_{ij}) = \frac{\exp(-\kappa r_{ij})}{r_{ij}}$. κ is the Debye length, and it is equal to $3.2\sqrt{[C]}\text{nm}^{-1}$, where $[C]$ is the concentration of monovalent ions (Molar) to be described implicitly by a Debye-Hückel potential. $[C]$ should be the sum of all monovalent ion concentrations, regardless of electric charge. Table 5.7 describes the available options.

5.8 SCM.jar

This section describes a Java application `SCM.jar` that computes the "Shadow" map, a general contact definition for capturing the dynamics of biomolecular folding and function. It is described in the literature here [13]. A contact map is a binary symmetric matrix that encodes which atom pairs are given attractive interactions in the SBM potential. In the context of a SBM, the native contact map should approximate the distribution of stabilizing enthalpy in the native state that is provided by short range interactions like van der Waals forces, hydrogen bonding, and salt bridges. Any long range interactions or nonlocal effects are taken into account in a mean field way through the native bias.

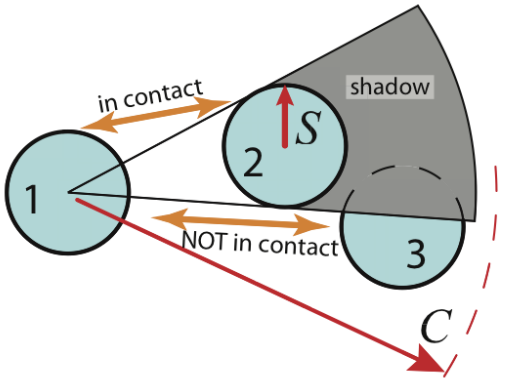


FIGURE 5.1: The Shadow contact map screening geometry. Only atoms within the cutoff distance C are considered. Atoms 1 and 2 are in contact because they are within C and have no intervening atom. To check if atoms 1 and 3 are in contact, one checks if atom 2 shadows atom 1 from atom 3. The three atoms are viewed in the plane, and all atoms are given the same shadowing radius S. Since a light shining from the center of atom 1 causes a shadow to be cast on atom 3, atoms 1 and 3 are not in contact. See Section 5.8.1.

Role of `SCM.jar` in SMOG 2

Internally SMOG 2 uses `SCM.jar` to compute contact maps. From the user's point of view the contact map can be of two types, all-atom or coarse-grained. An all-atom map returns the atoms that are in contact based on the Shadow definition. A coarse-grained map (e.g. to be used with the C_α model) is created from an all-atom map. The coarse-grained map consists of residue-level contacts. A residue-level contact exists if there is at least one atom-atom contact between two residues. This is why a PDB containing all heavy atoms is required by the tool. When coarse-graining SMOG 2 asks that the user provide an all-atom template in addition to the coarse-graining template that tells

SMOG 2 how to interpret the all-atom PDB in order to interface with `SCM.jar`. The actual command within the code is:

```
java $memoryMax -jar $ENV{SMOG_PATH}/src/tools/SCM.jar -g $groFile4SCM
     -ndec 4 -t $topFile -o $shadowFile -ch $ndxFile $SCMparams
```

The additional `$SCMparams` are set in the subroutine `setContactParams()`. This subroutine reads the attributes in the `Contact` tag of the `.sif` template file. The standalone tool is available within the SMOG 2 tools directory for users that want to create their own customized maps (also the source code is there). The rest of the chapter describes the basics of using the tool.

Locating `SCM.jar`: The jar should be located in $SMOG_PATH/src/tools.

Citing `SCM.jar`: The citation for `SCM.jar` is [13].

Running `SCM.jar`

Like any java application, no compilation is necessary, but a virtual machine is required; `SCM.jar` requires a sufficiently recent JRE. `SCM.jar` reads SMOG formatted Gromacs input files. **Important! The all (heavy) atom geometry must be used**, even if the output will be a coarse-grained residue-based map for a C_α model. The atomic coordinates are read in `.gro` format and the bond connectivity is read via a `.top` obtained from the SMOG webtool (or source distribution). The topology is required since bonded atoms shadow each other differently and since contacts are automatically discarded between two atoms if they share a bonded interaction (bond, angle, dihedral). At the command line, the basic syntax is

```
user$ java [-Xmx1000m] -jar SCM.jar -g grofile -t topfile -o outputName \
[--chain chainFile] [--default | -m {shadow,cutoff}]
```

`-Xmx1000m` assigns 1000 MB of RAM to the Java virtual machine heap. With large complexes (>1e5 atoms) the default heap allocation can run out which gives the following error:

```
java.lang.OutOfMemoryError: Java heap space
```

The output all-atom contact file format is

```
chain_i atom_i chain_j atom_j [distance]
```

and similarly, the output residue contact file format is

```
chain_i residue_i chain_j residue_j [distance]
```

Some examples

- Shadow map, atomic contacts, shadowing radius 1 Å and cutoff 6 Å (default sizes). See Figure 5.1 for definition of radius and cutoff. Add `--chain` if you have multiple chains, since the `.gro` format does not allow for chain information. Specify the chains file you get from your SMOG output.

  ```
  user$ java -jar SCM.jar -g protein.gro -t protein.top -o contactsOut
  --default [--chain chainsFile]
  ```

- Same as above, but including the correction such that the algorithm exactly follows the description in Figure 5.1 (see Section 5.8.1).

  ```
  user$ java -jar SCM.jar -g protein.gro -t protein.top -o contactsOut
  --default [--chain chainsFile] --correctedShadow
  ```

- Shadow map, atomic contacts, shadowing radius 2 Å and cutoff 4 Å.

  ```
  user$ java -jar SCM.jar -g protein.gro -t protein.top -o contactsOut
  -m shadow -c 4 -s 2 [--chain chainsFile]
  ```

- Cutoff map, atomic contacts, and cutoff 4 Å.

  ```
  user$ java -jar SCM.jar -g protein.gro -t protein.top -o contactsOut
  -m cutoff -c 4 [--chain chainsFile]
  ```

  ```
  - OR -
  ```

  ```
  user$ java -jar SCM.jar -g protein.gro -t protein.top -o contactsOut
  -s shadow -s 0 -c 4 [--chain chainsFile]
  ```

- Shadow map, residue contacts, default, include contact distances

  ```
  user$ java -jar SCM.jar -g protein.gro -t protein.top -o contactsOut
     --distances --coarse CA [--chain chainsFile]
  ```

- To calculate over a trajectory instead of a single structure, use `--multiple X`, where `X` is the number of frames in the trajectory `.gro` file. Assumes that the format of `proteinTraj.gro` is the same as the output of `trjconv`. This saves time relative to looping over many grofiles because the topology (and therefore the bonded list) is only read once.

```
user$ $GROMACS/trjconv -f traj.xtc -o proteinTraj.gro
```

```
user$ java -jar SCM.jar -g proteinTraj.gro -t protein.top -o contactMapsOut
--default --multiple 1000 [--chain chainsFile]
```

Some details of coarse-graining

The coarse-grained contact map returned is only strictly recommended for use with C_α models of proteins, and where the input PDB has an all-atom representation. For various modeling applications, it is desirable that the program not die with an error if the PDB doesn't only contain all-atom protein with each residue containing a CA atom. Therefore, the behavior is that the program will choose one atom from each residue to stand in as the representative coarse-grained position. It chooses, in order of preference: `CA`, `N1`, first atom in the residue. This really only matters for the `--distance` option.

Full configuration parameter list

The following will give a full list of configuration options:

```
user$ java -jar SCM.jar -help
```

Running `SCM.jar` through the webtool

On the webserver (http://smog-server.org/Shadow.html) one can build a shadow map from a SMOG formatted PDB file.

5.8.1 Corrected algorithm

From the beginning, SCM.jar has had a bug where `asin` was replaced by `atan` in the code calculating the tangents to the spheres (see Figure 5.1). This went undetected since the error is small when the distances between atoms is larger than the shadowing size, roughly 3% error in the angle for a shadowing atom 4 Å away. This works out to allow slightly more contacts. The option `--correctedShadow` (introduced in v2.3)

implements the exact algorithm of Figure 5.1, whereas SCM.jar without that switch continues to use the "buggy" form.

If the user wishes to use `--correctedShadow`, but wants to create maps as close to the "buggy" maps as possible, a good rule of thumb is to use a slightly smaller shadow size. For example, using `--correctedShadow -m shadow -s 0.975 -c 6` for the CI2 protein (1YPA.pdb), gives the same number of contacts (599) with 99.5% of the contacts identical.

5.9 `WHAM.jar`

This section only provides a small subset of WHAM.jar capabilities, where many (e.g. umbrella potential analysis, etc) are not described. We are working on organizing usage examples, and we hope to update this description soon. Until then, please reach out if you have questions.

The Java application `WHAM.jar` at its heart uses a well established algorithm described here [14]. `WHAM.jar` is tailored for the sort of analysis that is most often performed on SBMs, including free energy perturbation, thermal averaging of reaction coordinates and free energy as a function of (1 or 2) reaction coordinates. The basic operation involves providing `WHAM.jar` with a set of histograms from constant temperature simulations, which can vary in both temperatures and umbrella parameters, and a configuration file. Based on the options set in the configuration file (or at the command line) `WHAM.jar` will perform the appropriate analysis. The WHAM algorithm itself [14] provides an optimal density of states (Ω). From Ω many thermodynamic quantities of interest can be calculated.

Some powerful features of `WHAM.jar` are worth mentioning. As the system size grows, the energy grows, and this can lead to overflow of double precision floating point numbers (11 bit exponent, $2^{11} \approx 1000$) when exponentiating the energy. `WHAM.jar` uses a 32 bit exponent, allowing energies up to 10^9. The efficiency of SMOG models can allow for multiple umbrella potentials to be simultaneously employed and sufficiently sampled. `WHAM.jar` can reweight arbitrarily many umbrellas. However, note that the code currently only allows for the umbrella energy to be quadratic along the umbrella coordinate (but this can be easily changed). As more dimensions are added to the histogram, i.e. multiple reaction coordinates and umbrella coordinates, the time and memory requirements dramatically increase. `WHAM.jar` uses a sparse array data structure, which eliminates memory issues and allows arbitrary histogram bounds and steps to be used.

Additionally, Java Threads are used. Set `threads <int>` in the configuration file to make use of multiple cores.

Running `WHAM.jar`

Like any java application, no compilation is necessary, but a virtual machine is required; `WHAM.jar` requires JRE 1.6.0_29 or greater. At the command line, the basic syntax is

```
user$ java [-Xmx100m] -jar WHAM.jar --config configurationFile
```

`-Xmx100m` assigns 100 MB of RAM to the Java virtual machine heap. With large histograms (4 or 5 dimensions) the default heap allocation can run out which gives the following error:

```
java.lang.OutOfMemoryError: Java heap space
```

The error handling is often not very user friendly, so make sure to look at the formatting of examples if you are running into Java exceptions during runtime.

5.9.1 Configuration file

5.9.1.1 Example 1: Combining constant temperature runs

Find the example files at `$SMOG_DIR/examples/whamExamples/1`. This example introduces the basic use of `WHAM.jar` by reading histograms from 3 constant temperature runs (T=144.2,144.3,144.4), running WHAM, and outputting a specific heat (C_v diagram) and a free energy with respect to a reaction coordinate Q. Here, Q is the number of native contacts formed.

Each trajectory (i.e. thermodynamic state) gets its own file. The histogramming is controlled by the user. Reducing the number of bins speeds up the calculation, but reduces resolution of the output. Too many bins will result in a lack of data per bin and noise will dominate.

```
--------- start config file -------------------

kB 0.008314 # Boltzmann constant
tolerance .001 # when convergence is reached
overwriting # WHAM.jar is allowed to overwrite
            # existing files
```

```
threads 1   # increase to utilize multiple cores

##### define internal histograming #####
numDimensions 2 # space delimited file, two columns: Potential_energy and Q
# energy
numBins 144
start -88
step 2
# Q
numBins 100
start 0
step 2

##### trajectory files (each with 2 columns: Potential_energy and Q)
numFiles 3
name data/hist.144.2.in temp 144.2  # temp specifies the temperature of
                                    # this trajectory
name data/hist.144.3.in temp 144.3
name data/hist.144.4.in temp 144.4
```

- The Boltzmann constant (`kB`) relates the energy units used in the histogram files to the temperature units given with `temp`. The value shown here is appropriate for Gromacs and SMOG, where the temperature is the Gromacs temperature in Kelvin.
- The histogram files are designated with relative paths with respect to the directory where WHAM.jar is launched. Histogram files are space delimited columns with the following format:

  ```
  energy reaction_coord_1 [ more reaction coords ] [ umbrella coords ]
  ```

- The total span of the histogram is from `start` to `start+numBins*step`.
- `overwriting` is used in the example so that it doesn't die while running due to the prior existence of output files. One can remove this parameter in order to ensure that WHAM.jar doesn't overwrite some previous analysis.

To run WHAM add the following to the config file:

```
run_wham  # tells the program to compute density of states
dosFile dos # filename for density of states output
```

Only the histogram bins that have at least one data point are written. The density of states histogram columns are in the same order as the input with an additional column that is the density at the indicated bin. After the program computes the density of states (written to `dos`), there are routines that perform various integrals to obtain thermodynamic information.

For instance, to calculate the specific heat add the following to the config file:

```
run_cv # tells the program to compute specific head from dos
### run_cv parameters
startT 135 # start temp
deltaT 0.1 # print every 0.1 degree
ntemps 200 # for a total of 200 prints
run_cv_out cv # name of cv outputfile
```

- There will be one output file with five columns:

  ```
  temperature Cv enthalpy free_energy entropy
  ```

- $C_{\mathrm{v}}(T) = \langle E^2 \rangle - \langle E \rangle^2$

To compute $F(Q_1), S(Q_1), E(Q_1)$ at T from $\Omega(E, Q_1)$ or $F(Q_1, Q_2), S(Q_1, Q_2), E(Q_1, Q_2)$ from $\Omega(E, Q_1, Q_2)$ add the following to the config file:

```
run_free #tells the program to compute free energies
startTF 135 # start temp
deltaTF 1 # print every 1 degree
ntempsF 20 # for a total of 20 prints
run_free_out free/  # filename is ${run_free_out}${temp x 10}
#NOTE any output file can have directory structure
```

- Since the first column is always energy, the program knows the second column is the reaction coordinate.
- There will `ntempsF` files with four columns:

  ```
  reaction_coord_value free_energy enthalpy entropy
  ```

- $F(Q_1) = -kT \ln \Sigma_E \Omega(E, Q_1) \exp(-E/kT)$
- $E(Q_1) = \frac{\Sigma_E E \Omega(E,Q_1) \exp(-E/kT)}{\Sigma_{E,Q_1} \Omega(E,Q_1) \exp(-E/kT)}$

- $TS(Q_1) = F(Q_1) - E(Q_1)$
- If two reaction coordinates are given (i.e. `numDimensions = 3`), `run_free` assumes you want two dimension free energies, i.e. $F(Q_1, Q_2)$. Remember to add another histogram descriptor to the config file:

```
# Q2
numBins 100
start 0
step 2
```

To compute $\langle Q_1(T) \rangle$ from $\Omega(E, Q_1)$ or $\langle Q_1(Q_2) \rangle$ at T from $\Omega(E, Q_1, Q_2)$ add the following to the config file:

```
run_coord #tells the program to compute coordinate averages
startTC 135 # start temp
deltaTC 1 # print every 1 degree
ntempsC 20 # for a total of 20 prints
run_coord_out coord  # filename
```

- Since the first column is always energy, the program knows the second column is the reaction coordinate.
- There will be one output file with two columns:

```
temperature <Q(T)>
```

- $\langle Q_1(T) \rangle = \frac{\Sigma_E Q_1 \Omega(E,Q_1) \exp(-E/kT)}{\Sigma_E \Omega(E,Q_1) \exp(-E/kT)}$
- If two reaction coordinates are given (i.e. `numDimensions = 3`), `run_coord` assumes you want the expectation value of Q_1 (second column) as a function of Q_2 (third column), i.e. $\langle Q_1(Q_2) \rangle$. The temperature is taken as `startTF`. Remember to add another histogram descriptor to the config file:

```
# Q2
numBins 100
start 0
step 2
```

Chapter 6

Template-Based Approach

6.1 Introduction to templates

SMOG 2 offers increased versatility over SMOG v1 by shifting to a template-based approach for defining molecular structures. Each template allows for direct control of the structure-based energy function, which may include (but is not limited to) multi-resolution models and models that include non-specific interactions. The plug-and-play nature of the templates has the additional advantage of force field portability and easy sharing of user-created variations of structure-based potentials.

A single SMOG "template" is comprised of four required XML-formatted files, plus two optional ASCII files. XML was adopted because of its standardized formatting, ease of editability and readability, and there are widely available program modules to generate and parse XML files. Furthermore, XML allows for schemas, a content format restriction file, to which the template files must conform, which adds an additional layer of error checking capabilities. This chapter assumes that the user knows the basics of XML formatting. Users unfamiliar with XML formatting may want to check out the W3schools' website.

Table 6.1 summarizes the purpose of each template file. In Chapter 7, we show how to add new residues to the template files for an all-atom structure-based model.

6.2 SMOG 2 Templates

As discussed in Chapter 3, if one is not using a default model, then the user must provide a template folder as input when calling SMOG 2. The template folder contains all files needed to define the rules for constructing a structure-based model. Within

File	Purpose
Biomolecular Information File (.bif)	defines the structure of biomolecules to be supported
Setting Information File (.sif)	defines interaction function declarations
Bond File (.b)	defines bonded interactions between atoms
Nonbond File (.nb)	defines non-bonded interactions between atoms
extras file (.extras file; optional)	used to add static content to force fields
ions file (.ions.def file; optional)	used to by smog_ions when adding ions

TABLE 6.1: Descriptions of SMOG 2 template files

this directory, four files are mandatory (.sif, .bif, .b, .nb files), while three are optional (extras, ions.def and citation file). A template folder can only contain **one** of each file type. If your template folder contains more than one file of a specific type, the program will exit with an error. Each file contains unique information, as described below.

6.2.1 Biomolecular Information File (.bif)

The Biomolecular Information File (from here on called .bif) defines the covalent structure of all residues described by a particular force field. Each residue is defined in the .bif file by declaring all the atoms in that residue, the bonds between the atoms and the improper dihedrals between the atoms.

6.2.1.1 Residues

Each residue is individually defined between the `<residues>` and `</residues>` tags. As an example, the text below shows how one would define the residue ALA, which contains 5 atoms.

```
1               <residue name="ALA" residueType="amino" atomCount="5">
2                       <atoms>
3                               <atom bType="B_1" nbType="NB_1" pairType="P_1">N</atom>
4                               <atom bType="B_1" nbType="NB_1" pairType="P_1">CA</atom>
5                               <atom bType="B_1" nbType="NB_1" pairType="P_1">C</atom>
6                               <atom bType="B_1" nbType="NB_1" pairType="P_1" charge="-1.0">O</atom>
7                               <atom bType="B_1" nbType="NB_1" pairType="P_1">CB</atom>
8                       </atoms>
9                       <bonds>
10      <!--BACKBONE-->
11                              <bond energyGroup="bb_a">
```

```
12                                      <atom>N</atom>
13                                      <atom>CA</atom>
14                              </bond>
15                              <bond energyGroup="bb_a">
16                                      <atom>CA</atom>
17                                      <atom>C</atom>
18                              </bond>
19                              <bond energyGroup="bb_a">
20                                      <atom>C</atom>
21                                      <atom>O</atom>
22                              </bond>
23          <!--FUNCTIONAL GROUP-->
24                              <bond energyGroup="sc_a">
25                                      <atom>CA</atom>
26                                      <atom>CB</atom>
27                              </bond>
28                      </bonds>
29                      <impropers>
30                              <improper>
31                                      <atom>CB</atom>
32                                      <atom>CA</atom>
33                                      <atom>C</atom>
34                                      <atom>N</atom>
35                              </improper>
36                      </impropers>
37              </residue>
```

LISTING 6.1: Residue section of .bif file

The attribute *name* is the name of the residue, as used in your PDB file. The attribute residue *residueType* is the type of residue, in this case, an amino acid residue. Finally the optional attribute *atomCount* allows the user to explicitly set the total number of atoms to be counted for normalizing energies. That is, the total atom count is used in the energetic scaling procedure of dihedrals and contact energies, as described in Appendix A. This feature is useful when including many copies of a ligand in your system, since the energetic normalization should only be based on the protein, or RNA, and not the multiply-copied ligands. In such a scenario, the user would set *atomCount* to 0. If *atomCount* is not defined, SMOG 2 automatically uses the total number of atoms listed under the `<atoms>` tag.

The `<atoms>` tag declares all the atoms in the residue. Note that all the atoms within a specific residue in your PDB **must** be listed here. If the PDB and .bif are not consistent, the program will terminate with an error. This is the reason that the default templates differentiate between the C-terminal and non-terminal protein residues (See Chapter 3). Each atom has a bond type *bType*, a non-bond type *nbType* and a pair type *pairType*. The bond type attribute is used in the generation of the bonded interactions (bonds, angles, and dihedrals). Likewise, the non-bond type attribute is used in the generation of the non-bonded interactions. The pairType attribute is used in the generation of

contact interactions (6-12, 10-12 or Gaussian interactions). In the example above, we also show the optional *charge* attribute, which allows the user to specify the charge of an atom within a residue. Using *charge* within an *atom* declaration will supersede any charge assignment based on *nbType* (see below).

The `<bonds>` tag contains all the bonds that should be present in the residue. Each bond in a residue is listed under the `<bond>` tag. The atom names must match those listed in the `<atoms>` field. The bond tag also has an attribute called *energyGroup* that allows for one to define heterogeneous energetics in the system. The energy group attribute is used in conjunction with the bond types to determine the dihedral interaction. Using the bonds declared here, the program dynamically identifies and calculates all angles and dihedrals that can be defined for the molecule.

The `<impropers>` tag contains all the improper dihedral angles in the biomolecule. The tag `<improper></improper>` holds four atom tags. The order of the four atoms defines a specific improper dihedral within a residue. This feature is used to add dihedrals that cannot be determined based on bond geometry.

6.2.1.2 Connections

In addition to defining a residue, the .bif file is also used to define how sequential residues are covalently connected. Listing 6.2.1.2 shows how two residues of type amino are covalently linked. The attribute *residueType1* and *residueType2* declares how a residue of type residueType1 at position n should be connected to a residue of type residueType2 at position n+1. The residue types are matched based on the residue definitions. Much of the structure of the connection element is similar to that of the residue element. There is a single bond, whereby the first atom belongs to the nth residue and the second atom belongs to the (n+1)th residue. You can also define a single improper dihedral, though this is not a requirement component of all connection definitions. In the context of impropers, the special character suffix "+" is used to declare atoms that belong to the (n+1)th residue. In listing 6.2, the N atom belongs to the (n+1)th residue.

Note: If you do not want to covalently link residues that are listed sequentially in the PDB file, then you should add the attribute *connect=no* to the definition of the first residue type within the *residue* declaration. This could be helpful if you are modeling ions, where each ion is listed as a single residue, but a block of ions appear in the PDB file between TER lines.

```
1       <connections>
2               <!-- AMINO/AMINO -->
```

```
3               <connection name="amino-amino" residueType1="amino" residueType2="amino">
4                       <bond energyGroup="r_a" >
5                               <atom>C</atom>
6                               <atom>N</atom>
7                       </bond>
8
9                       <improper>
10                              <atom>O</atom>
11                              <atom>CA</atom>
12                              <atom>C</atom>
13                              <atom>N+</atom>
14                      </improper>
15              </connection>
16        </connections>
```

LISTING 6.2: Connection section of .bif file

6.2.2 Setting Information File (.sif)

While the .bif file is used to define the covalent geometry of each residue, the Setting Information File (.sif) is used to control the distribution and allowable functional forms of the energetic terms, which includes the inter-dihedral dihedral ratios, contact-to-dihedral ratios, contact map settings and function declarations.

6.2.2.1 Functions

The `<functions>` tag should list all the functions that the model will use.

```
1  <functions>
2          <function name="bond_harmonic" directive="bonds"/>
3          <function name="bond_type6" directive="bonds"/>
4          <function name="bond_bytype" directive="bonds"/>
5          <function name="angle_harmonic" directive="angles"/>
6          <function name="angle_free" directive="angles"/>
7          <function name="angle_bytype" directive="angles"/>
8          <function name="dihedral_cosine" directive="dihedrals"/>
9          <function name="dihedral_harmonic" directive="dihedrals"/>
10         <function name="dihedral_free" directive="dihedrals"/>
11         <function name="dihedral_bytype" directive="dihedrals"/>
12         <function name="contact_1" directive="pairs" exclusions="1"/>
13         <function name="contact_2" directive="pairs" exclusions="1"/>
14         <function name="contact_gaussian" directive="pairs" exclusions="1"/>
15         <function name="contact_free" directive="pairs" exclusions="0"/>
16 </functions>
```

LISTING 6.3: Example of functions section of a .sif file

All functions that will be used in the model should be listed under the `<functions>` tag. These function names are mapped to specific subroutines in `src/smogv2`. While

functions available in Gromacs are static, the OpenSMOG option may be used to apply user-defined potentials when using OpenMM. All supported interactions are listed in Table 6.2.

The function tag has two attributes, `directive` and `exclusions`. `directive` takes a string and specifies the topology directive under which to write out the interaction. `exclusions` is boolean and specifies whether to write out the atom pair additionally under the `[ exclusions ]` directive (when not using OpenSMOG). This means that the atom pair is not included in the non-specific excluded volume interaction list. This is because the pair interaction has its own excluded volume part and it shouldn't be double counted. Since there can be interactions defined that do not include excluded volume, we include the option for flexibility. Note that when using the -OpenSMOG option, exclusions are not written in the top file, but are instead indicated by the "exclusions" tag in the output XML files.

SMOG 2 input		Potential energy	Gromacs .top		OpenSMOG
Name	**Args. (units)**		**declaration**	**ftype**	**supported?**
bond_harmonic	r_0 (nm), ϵ_r (ϵ/nm^2)	$\frac{\epsilon_\mathrm{r}}{2}(r-r_0)^2$	[bonds]	1	yes
bond_type6[a]	r_0 (nm), ϵ_r (ϵ/nm^2)	$\frac{\epsilon_\mathrm{r}}{2}(r-r_0)^2$	[bonds]	6	yes
angle_harmonic	θ_0 (deg), ϵ_θ (ϵ/rad^2)	$\frac{\epsilon_\theta}{2}(\theta-\theta_0)^2$	[angles]	1	yes
angle_free	N/A	N/A			yes
dihedral_harmonic	ϕ_0 (deg), ϵ_χ (ϵ/rad^2)	$\frac{\epsilon_\chi}{2}(\phi-\phi_0)^2$	[dihedrals]	2	yes
dihedral_cosine[b]	ϕ_0 (deg), ϵ_D (ϵ), n	$\epsilon_D[1-\cos(n(\phi-\phi_0))]$	[dihedrals]	1	yes
dihedral_ncos[b]	ϕ_0 (deg), ϵ_D (ϵ), n	$\epsilon_D[1-\cos(n\phi-\phi_0)]$	[dihedrals]	1	yes
dihedral_pcos[b]	ϕ_0 (deg), ϵ_D (ϵ), n	$\epsilon_D[1+\cos(n\phi-\phi_0)]$	[dihedrals]	1	yes
dihedral_cosine4[b]	ϕ_0 (deg), ϵ_D (ϵ), n	$\epsilon_D[1-\cos(n(\phi-\phi_0))]$	[dihedrals]	4	yes
dihedral_ncos4[b]	ϕ_0 (deg), ϵ_D (ϵ), n	$\epsilon_D[1-\cos(n\phi-\phi_0)]$	[dihedrals]	4	yes
dihedral_pcos4[b]	ϕ_0 (deg), ϵ_D (ϵ), n	$\epsilon_D[1+\cos(n\phi-\phi_0)]$	[dihedrals]	4	yes
dihedral_free	N/A	N/A			yes
contact_1[c]	N, M, σ (nm), ϵ_C (ϵ)		[pairs]	1	yes
contact_2[d]	f(σ), g(σ), σ (nm), ϵ_C (ϵ)		[pairs]	1	no
contact_gaussian[e]	ϵ_C (ϵ), r_NC^{12} (nm^{12}), σ (nm), r_0 (nm)	See appendix A.3.5.1	[pairs]	6	yes
contact_free	don't assign contacts between specific sets of atoms				yes
User-defined angles	user-defined	See section 4.1.2	N/A	N/A	only
User-defined dihedrals	user-defined	See section 4.1.2	N/A	N/A	only
User-defined contacts	user-defined	See section 4.1.2	N/A	N/A	only

The energy unit ϵ for all parameters is the reduced energy unit (see A.4), which is labeled kJ/mol in Gromacs.
[a] Can be used as a contact potential in .nb for applications like restraining bound ions or elastic network models. Identical to bond_harmonic, except it is not treated as a chemical bond.
[b] When using Gromacs, the parameters in a top file are given for the functional form of $\epsilon_D[1+\cos(n\phi-\phi_0)]$.
[c] Minimum at native distance if using ? for σ.
[d] Minimum at native distance if using ? for σ. When using Gromacs, one needs to include a table file with `-tablep` with `mdrun`.

TABLE 6.2: Potential energy functions available in SMOG 2

6.2.2.2 Declaring a contact type

There are three ways to declare Lennard-Jones-style potentials for contacts (See Table 6.2): contact_1 and contact_2, as well as through a user-defined potential.

The form of the potential for contact_1 is:

$$V_{\text{contact}} = \frac{A}{r^M} - \frac{B}{r^N} \tag{6.1}$$

A and B are automatically evaluated, such that the minimum is at distance σ and depth ϵ. Distances are calculated based on the input PDB if "?" is given in the σ field.

The form of the potential for contact_2 is:

$$V_{\text{contact}} = \epsilon \left(f(\sigma)G(r) - g(\sigma)H(r) \right) \tag{6.2}$$

$f(\sigma)$ and $g(\sigma)$ are expressions that are evaluated using the native distance if "?" is given for σ. The functions $G(r)$ and $H(r)$ are typically provided to Gromacs via table files (See Section 5.7).

6.2.2.3 Defining the contact map

For any given biomolecular structure, one needs to define a set of non-local interactions that are stabilizing. These interactions are collectively called the "contact map". The definition of the contact map is probably the most important single element of any SMOG model. Accordingly, SMOG 2 allows for a range of strategies to be employed. One approach is for the user to provide their own map (see section 3.2.6). However, it is more common that SMOG 2 is used to generate the contact map for you. This is accomplished with the `Contacts` element in the sif file. Below are some examples for how contact maps may be defined in SMOG 2.

The example below will tell SMOG 2 to use the Shadow Contact Map algorithm [13] with a maximum contact distance of 6Å with a shadowing radius of 1Å for atoms that are not bonded, and a radius of 0.5Å for atoms that are bonded.

```
1    <Contacts method="shadow" contactDistance="6" shadowRadius="1" shadowRadiusBonded="0.5"/>
```

Listing 6.4: Calculating a shadow map

This example would simply use a cutoff distance of 6 Å.

```
1    <Contacts method="cutoff" contactDistance="6"/>
```

Listing 6.5: Calculating a cutoff map

It is also possible to tell SMOG 2 to rescale the weights of contacts between certain types of atoms. For example:

```
<Contacts method="cutoff" contactDistance="4" >
<contactScaling name="stackingScale" residueType1="nucleic" residueType2="nucleic" scale="0.3"
deltaMin="1" deltaMax="1"
atomList="N1 N2 N3 N4 N5 N6 N7 N8 N9 C1 C2 C3 C4 C5 C6 C7 C8 C9 O1 O2 O3 O4 O5 O6 O7 O8 O9 S10
C11 C12 C13 C14 C15 C16"/>
</Contacts>
```

LISTING 6.6: Calculating a cutoff map with rescaled interactions

In this example, a cutoff map would be calculated, but interactions between adjacent residues (set by deltaMin and deltaMax) that are in the atomList group will be rescaled by 0.3.

6.2.2.4 Group Settings

Structure-based models have two classes of energy groups: contact groups and dihedral groups. Each contact group represents a collection of contacts, and each dihedral group represents a collection of dihedrals. These energy groups are used for energetic scaling of interaction strengths.

$$U_{\mathrm{AA}} = \ldots + \sum_{\text{backbone}} \epsilon_{\mathrm{BB}} F_D(\phi) + \sum_{\text{sidechain}} \epsilon_{\mathrm{SC}} F_{\mathrm{D}}(\phi) \tag{6.3}$$

$$+ \sum_{\text{contacts}} \epsilon_{\mathrm{C}} F_{\text{contacts}}(r) + \ldots \tag{6.4}$$

Shown above are the dihedral and contact terms of the all-atom potential. Shown below are the energetic scaling factors for dihedrals and contacts, and their respective attributes under the .sif file.

$$\frac{\epsilon_{\mathrm{BB}}}{\epsilon_{\mathrm{SC}}} = \frac{\text{intraRelativeStrength bb}}{\text{intraRelativeStrength sc}} \tag{6.5}$$

$$\sum \epsilon_{\mathrm{BB}} + \sum \epsilon_{\mathrm{SC}} + \sum \epsilon_{\mathrm{C}} = \text{total non-ligand atoms} \tag{6.6}$$

$$\frac{\sum \epsilon_{\mathrm{BB}} + \sum \epsilon_{\mathrm{SC}}}{\sum \epsilon_{\mathrm{C}}} = \frac{\text{dihedrals groupRatio}}{\text{contacts groupRatio}} \tag{6.7}$$

The program automatically calculates the total number of atoms used in the energetic scaling (ligands are excluded in the default models). Although the scaling equations shown above are limited to residue types with only two dihedral types (backbone and sidechain dihedrals) and a single contact type, the program allows for scaling equations to be generalized to more than two dihedral types and more than one contact type. The energy group ratios are contained within the `<Groups>` tag.

```
1        <energyGroup name="bb_n" residueType="nucleic" intraRelativeStrength="1" normalize="1"/>
2        <energyGroup name="sc_n" residueType="nucleic" intraRelativeStrength="1" normalize="1"/>
3        <energyGroup name="pr_n" residueType="nucleic" normalize="0"/>
4        <energyGroup name="ip_n" residueType="nucleic" normalize="0"/>
5        <energyGroup name="r_n"  residueType="nucleic" normalize="0"/>
6        <contactGroup name="c" intraRelativeStrength="1" normalize="1"/>
7        <groupRatios contacts="2" dihedrals="1"/>
```

LISTING 6.7: Energy group section of .sif file

The two classes of energy group ratios, dihedral and contact ratios, are determined by the `<energyGroup>` and `<contactGroup>` tags respectively. The *residueType* attribute is used to designate the residue type the scaling factors of a particular energy group is used for. The *name* attribute is the label for the energy group. The *name* attribute used here is matched to the *energyGroup* attribute under `<bond>` tag in the .bif file. The name of a particular contact energy group will be used later when declaring contact interaction functions in the subsequent section of this chapter.

The *normalize* attribute for each energy group is a boolean (1 or 0), and it is used to determine if a particular energy group should be included in energy normalization (see equation 6.6). For the all-atom model, the dihedral group with the name "pr_n" (which represents the nucleic planar rigid dihedrals) has a normalize option set to 0, indicating that planar dihedrals in nucleic acids will not be part of the normalization. In contrast, in the all-atom model, sidechain dihedrals are normalized, as are backbone dihedrals and contact energies. Accordingly, those energy groups have the normalize option set to 1. The *intraRelativeStrength* attribute is the relative ratio of stabilizing energy within the different class of energy group for a particular residue (equation 6.5).

Finally we use the `<groupRatios>` tag to set the energy partition between the two classes of energy groups according to equation 6.7.

6.2.3 Bond File (.b)

The .b file is used to define all bonded parameters, which includes bonds, angles, and dihedral.

We first discuss how to define a basic bond interaction for the all-atom model.

```
1  <!-- BONDS -->
2  <bonds>
3          <bond func="bond_harmonic(?,10000)">
4          <bType>*</bType>
5          <bType>*</bType>
6          </bond>
7          <bond func="bond_type6(?,200)">
8          <bType>*</bType>
9          <bType>MG</bType>
10         </bond>
11 </bonds>
```

LISTING 6.8: Bonds section of .b file

Recall that each atom is given a *bType* when they are declared in the .bif file. Given a particular bonded interaction (bonds, angles, dihedrals, impropers), the functional form for a bonded interaction is assigned by matching the combination of the *bType*s in that interaction. The first `<bond>` tag (line 3 in Listing 6.8) is used to assign a function called bond_harmonic to two bonded atoms of any type. Since the vanilla model has only B_1 atoms, the *s could be replaced with B_1, and the same bonds would be assigned. Recall under Listing 6.3, line 2, we defined bond_harmonic() as a type 1 function under the bonds directive (harmonic bond function). The input parameters for bond_harmonic() are r_0 and ϵ_{bond}. In this case we use the special character "?" to tell SMOG to calculate the native bond length (r_0) from the PDB structure file. You can instead also give a specific value for the bond distance. This feature can be useful when adding nonspecific/empirical terms (e.g. an AMBER/CHARMM backbone) to the potential. However, the extras file may also be used to define non-specific parameters.

The *bType* attribute here can take either an exact bond type or a special wildcard "*" character that matches to all available *bType*s. For the case of the all-atom model, since all the *bType* is identical, we can instead also defined the bond interaction as shown under the `<bond>` tag in line 8 of Listing 6.8. **Please note that the program will assign the interaction that most closely matches the *bType*s of a given atom pair.** One needs to be careful not to declare interactions that conflict with one another. For example, if your system contains a bond between atoms of bType B_1 and B_2, and bond definitions are only given for bTypes B_1-* and B_2-*, then SMOG would not know which function to apply, and it will exit with an error. However, if one also explicitly defines a bond function for B_1-B_2 pairs, that bond would take priority.

The angle interaction follows a similar form as bonds, but instead of expecting two *bType* attributes, it requires three. The *bType* attribute in this case is symmetric to the central *bType*. When matching bond angles, the angle definition that matches the most atoms identically will be used. Again, if an equal number of atoms match in multiple angle definitions, there would be ambiguity, and SMOG will quit.

Declaring dihedral potentials is handled in a manner that is similar to bond and angles, through there is one important difference. That is, in addition to matching the *bType*s, the central bond must have the same value for the *energyGroup*. As an example, we will show the "dihedral_cosine" function in listing 6.9, where this function is classified under the dihedral directive with function type 1 in the .sif file (Listing 6.3). The arguments for this function are of the form $(\phi_0, \epsilon_d, \text{mult})$ (see Tab. 6.2). As introduced earlier, the special character "?" tells SMOG 2 to calculate the native value from the PDB structure file. In this example, we tell SMOG 2 to calculate the dihedral angle for all dihedral interactions that involve the *bType* combination *-*-*-*, where the central two atoms are connected by a bond with an *energyGroup* of "bb_n". By using the "?" argument, along with a multiplicity factor (third argument), we tell the program to multiply all the ϕ_0 values by the multiplicity factor. More generally, one may provide any function for the angle calculation. For example, if you were to use (?*2-1), the angle would be evaluated as: native angle, times 2 and minus 1.

$$F_D(\phi) = (1 - \cos(\phi - \phi_0)) + 0.5(1 - \cos(3(\phi - \phi_0))) \tag{6.8}$$

Equation 6.8 shown above is the dihedral interaction function used in the all-atom model. The code Listing 6.9 shows how to assign $F_D(\phi)$ to all dihedrals of *energyGroup* "bb_n" in a system.

```
1 <dihedral func="dihedral_cosine(?,1,1)+dihedral_cosine(?,0.5,3)" energyGroup="bb_n">
2         <bType>*</bType>
3         <bType>*</bType>
4         <bType>*</bType>
5         <bType>*</bType>
6 </dihedral>
```

Listing 6.9: Dihedral section of .b file

As shown in the example, multiple functions can be applied to the same dihedral angle by including a sum of functions (e.g. func="f(..)+g(..)+h(..)"). Currently, only sums of functions may be given in the .b file. However, subtractions can be introduced by assigning a negative weight to an interaction.

The special keyword "?" has limitations in where it can be used. For bonds, angles, and dihedrals it can only be used for the first input parameter to a function.

6.2.3.1 How SMOG matches dihedral parameters to specific dihedral angles

As with bonds and angles, it is quite easy to declare multiple *bType* sequences that will match to the same atoms in a system. For example, if you define a dihedral function for B_1-B_2-*-*, as well as B_1-*-*-* for the same *energyGroup*, then a dihedral in your system between atoms (B_1,B_2,B_1,B_1) would match both, unless the central bond was of a different *energyGroup*.

To determine which dihedral function should be applied to a given dihedral in your system, a scoring function S is calculated for all dihedral definitions, and the highest-scoring function is applied in your model. S is defined as 2 times the number of exactly matching *bType*s, plus the number of wildcard matches. If (at least) one of the atom positions does not have an exact match, nor a wildcard in that position, then $S = 0$. Also, if the *energyGroup* of the central bond in your structure is different from the *energyGroup* of the dihedral declaration, then $S = 0$, which enforces that the bond and dihedral are of the same *energyGroup*. The dihedral with the largest value of S is used. If more than one dihedral has the same value of S, or if $S = 0$, an error is thrown.

In the example above, $S = 6$ for the first definition and $S = 5$ for the second. Accordingly, the B_1-B_2-*-* definition would take priority. If two dihedrals have equal S values (ambiguous assignment), or $S = 0$ for all dihedral functions (i.e. no matches), then the program will quit with an error.

6.2.4 Nonbond File (.nb)

The .nb file is used to define non-bonded interactions, including native contacts and non-specific interactions.

```
1  <!-- DEFAULTS -->
2  <defaults gen-pairs="0" nbfunc="1" gmx-combination-rule="1" fudgeLJ="1" fudgeQQ="1"/>
3  <!-- GENERAL NONBONDS -->
4  <nonbond mass="1.00"  charge="0.000" ptype="A"  c6="0.0"  c12="5.96046e-9">
5          <nbType>NB_1</nbType>
6  </nonbond>
7  <!-- CONTACTS -->
8  <contact func="contact_1(6,12,?,energynorm)" contactGroup="c">
9          <pairType>*</pairType>
10         <pairType>*</pairType>
11 </contact>
```

Listing 6.10: Contacts section of .nb file

Listing 6.10 shows how to define non-bonded and native contact parameters. The *nbType* value defined for each atom in the .bif is matched to the nonbond declaration.

Note: Unless you are using the -OpenSMOG option, the contacts will be written under the `[ pairs ]` directive of the topology file. While this directive was originally intended for use with 1-4 pair interactions, we have adopted it to include contacts in SMOG models.

For structure-based models, a non-bonded interaction is a volume exclusion interaction usually defined as the Lennard-Jones 6-12 term. This is specified with the `nbfunc` tag in the `defaults` element in the .nb file. In addition, the parameters may be written in the .top file using the c6-c12 convention (`gmx-combination-rule=1`), or $\epsilon - \sigma$ convention (`gmx-combination-rule=2`). When using `gmx-combination-rule=2`, SMOG 2 will write values of σ in the `atomtypes` and `pairs` sections that preserve the intended SMOG functional form (Eq. 6.1), even though Gromacs interprets σ in terms of a traditional LJ function ($4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$). Gromacs generates these interactions using the information provided in the `[ defaults ]` and the `[ atomtypes ]` sections.

The non-bonded interaction declaration in SMOG 2 contains the atom attributes: mass, charge and atom type, as well as the explicit c6 and c12 terms for the Lennard-Jones function. If one wanted to include non-specific attractive interactions between atoms, then a non-zero value should be given for the c6 parameter.

You can also define fudge factors for 1-4 LJ and 1-4 Coulomb terms. None of the defaults are required to appear in the templates. If values are not defined, then SMOG 2 will assign `gen-pairs=0`, `gmx-combination-rule=1`, `nbfunc=1`, `fudgeLJ=1` and `fudgeQQ=1`.

A contact function declaration includes the additional *contactGroup* attribute, which is used to map the function to specific groups of contacts. For example, in Listing 6.7, the contact group c was declared with *intraRelativeStrength*=1 and *normalize* option set to 1 (true).

In code Listing 6.10, "?" marks indicate that the contact parameters c6, and c12 would be calculated automatically by the program using distances found in the PDB structure. `energynorm` tells SMOG 2 to calculate the value of ϵ_C (i.e. normalized) via the scaling equations. If one uses N-M functions in Gromacs, then table files have to be included (see Section 4.2.2 for syntax and details). Table files are not needed when running in OpenSMOG in openMM.

6.2.5 Extras file

There are many times when one may want to include static information in a top file. For example, you may want to use the bondtypes directive. For this type of information, one needs to include an "extras" file. This file may be called "extras" or it may have the suffix "extras". For an example on how to use this file, see Section 8.2.

6.2.6 ions.def file

When using smog_ions, one can either give the exact ion parameters on the command line, or you may give a template directory. In the latter case, smog_ions will look for a file called "ions.def" or a file that has the suffix "ions.def". If this file is found, then smog_ion will directly add the relevant terms to the `atomtypes` section of the top file.

```
1 MG      1       2       5.96046e-09     0
2 K       1       1       5.96046e-09     0
3 CL      1       -1      5.96046e-09     0
```

LISTING 6.11: example ions.def file. Format: ion name, mass, charge, c12, c6

Chapter 7

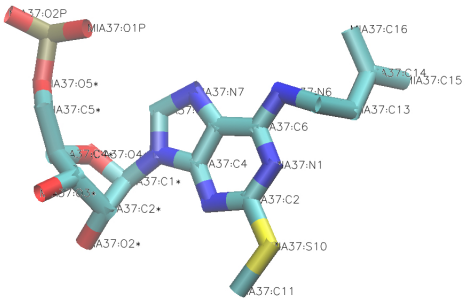
Adding a new residue

This chapter provides a step-by-step tutorial on how to add a new residue type using SMOG 2 template files. Rather than introduce a new residue, we will go through the declarations necessary the modified RNA residue MIA, which is now part of the default all-atom model.

The residue 2-methylthio-N6 isopentenyl adenosine (MIA) is a modified nucleic acid residue that is present in many RNA structures. It is nearly identical to Adenine, though there are a few additional atoms.

7.1 Step 1 - Examine the molecular structure

Make sure to have the correct chemical structure of your molecule. A useful method is to visually inspect it with a molecular visualization program (e.g. VMD or PyMOL):



7.2 Step 2 - Create a new All-Atom template directory

Since we're going to explicitly define each atom in the MIA residue, the example here will use the all-atom model. When adding a new residue, you can either copy an existing set of templates, or directly modify the default models. However, if you modify the default models, you should re-run smog-check to ensure that the baseline models were not inadvertently altered.

7.3 Step 3 - Define a new residue

As mentioned in Chapter 6, the biomolecular information file (.bif) defines the structure of all biomolecules in your system. Here we will define the residue information by declaring all of the atoms, bonds and improper dihedrals within the residue.

7.3.1 Place the new residue tag in the .bif file

As you get acquainted with the `AA-whitford09.bif` file structure (the default templates), you'll find that the residues appear grouped together according to their type: ligands, amino and nucleic residues. The residue type of MIA is nucleic, so it is added for convenience next to the existing nucleic residues. The `<residue>` tag encapsulates all of the residue information.

```
2945        <!-- NUCLEIC RESIDUES -->
2946
2947  <!--2-methylthio-N6 isopentenyl adenosine-->
2948                <residue name="MIA" residueType="nucleic">
2949                        <atoms>
2950                        </atoms>
2951                        <bonds>
2952                        </bonds>
2953                        <impropers>
2954                        </impropers>
2955                </residue>
2956
2957  <!--RNA A-->
2958                <residue name="A" residueType="nucleic">
```

LISTING 7.1: Nucleic residue section of .bif file

Keep in mind that the attribute *name* should match the residue name in the PDB file.

7.3.2 List all of the atoms in the residue

List all of the atom names in your residue as they appear in your PDB. The `<atoms>` tag encapsulates all the atoms in the biomolecule.

```
2945        <!-- NUCLEIC RESIDUES -->
2946
2947 <!--2-methylthio-N6 isopentenyl adenosine-->
2948                 <residue name="MIA" residueType="nucleic">
2949                         <atoms>
2950                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">P</atom>
2951                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">O1P</atom>
2952                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">O2P</atom>
2953                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">O5*</atom>
2954                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C5*</atom>
2955                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C4*</atom>
2956                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">O4*</atom>
2957                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C3*</atom>
2958                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">O3*</atom>
2959                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C2*</atom>
2960                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">O2*</atom>
2961                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C1*</atom>
2962                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">N9</atom>
2963                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C8</atom>
2964                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">N7</atom>
2965                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C5</atom>
2966                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C6</atom>
2967                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">N6</atom>
2968                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">N1</atom>
2969                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C2</atom>
2970                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">N3</atom>
2971                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C4</atom>
2972                                 <atom bType="B_1" nbType="NB_2" pairType="P_1">S10</atom>
2973                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C11</atom>
2974                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C12</atom>
2975                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C13</atom>
2976                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C14</atom>
2977                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C15</atom>
2978                                 <atom bType="B_1" nbType="NB_1" pairType="P_1">C16</atom>
2979                         </atoms>
2980                         <bonds>
2981                         </bonds>
2982                         <impropers>
2983                         </impropers>
2984                 </residue>
2985
2986 <!--RNA A-->
```

Listing 7.2: Adding the atoms section to the residue structure

In this example, as in the default models, all bonded interactions (bonds, angles and dihedrals) are defined the same for all atoms. Therefore, only one atom group needs to be defined. The bond type (*bType*) is `B_1` for all atoms. The contact interactions

pairType=P_1 are also defined to be the same for all atoms. However, changing the mass of a specific atom, such as sulfur (S10) in our example, will require a different non-bonded definition, since there will be a different excluded volume term. In this example, the new nbType NB_2 will have to be defined in the .nb file.

7.3.3 List all of the bonds

The chemical structure should tell you how atoms are covalently linked. Inspect those bonds and add them to the <bonds> section. The <bonds> tag encapsulates all the bonds in the biomolecule.

```
2980                    <bonds>
2981                <!--BACKBONE-->
2982                                <bond energyGroup="bb_n">
2983                                        <atom>P</atom>
2984                                        <atom>O1P</atom>
2985                                </bond>
2986                                <bond energyGroup="bb_n">
2987                                        <atom>P</atom>
2988                                        <atom>O2P</atom>
2989                                </bond>
2990                                <bond energyGroup="bb_n">
2991                                        <atom>P</atom>
2992                                        <atom>O5*</atom>
2993                                </bond>
2994                                <bond energyGroup="bb_n">
2995                                        <atom>O5*</atom>
2996                                        <atom>C5*</atom>
2997                                </bond>
2998                                <bond energyGroup="bb_n">
2999                                        <atom>C5*</atom>
3000                                        <atom>C4*</atom>
3001                                </bond>
3002                                <bond energyGroup="bb_n">
3003                                        <atom>C4*</atom>
3004                                        <atom>O4*</atom>
3005                                </bond>
3006                                <bond energyGroup="bb_n">
3007                                        <atom>C4*</atom>
3008                                        <atom>C3*</atom>
3009                                </bond>
3010                                <bond energyGroup="bb_n">
3011                                        <atom>C3*</atom>
3012                                        <atom>O3*</atom>
3013                                </bond>
3014                                <bond energyGroup="bb_n">
3015                                        <atom>C3*</atom>
3016                                        <atom>C2*</atom>
3017                                </bond>
3018                                <bond energyGroup="bb_n">
3019                                        <atom>C2*</atom>
```

```
3020                                    <atom>O2*</atom>
3021                            </bond>
3022                            <bond energyGroup="bb_n">
3023                                    <atom>C2*</atom>
3024                                    <atom>C1*</atom>
3025                            </bond>
3026                            <bond energyGroup="bb_n">
3027                                    <atom>C1*</atom>
3028                                    <atom>O4*</atom>
3029                            </bond>
3030                    <!--FUNCTIONAL GROUP-->
3031                                    <bond energyGroup="sc_n">
3032                                            <atom>C1*</atom>
3033                                            <atom>N9</atom>
3034                                    </bond>
3035                                    <bond energyGroup="pr_n">
3036                                            <atom>N9</atom>
3037                                            <atom>C8</atom>
3038                                    </bond>
3039                                    <bond energyGroup="pr_n">
3040                                            <atom>C8</atom>
3041                                            <atom>N7</atom>
3042                                    </bond>
3043                                    <bond energyGroup="pr_n">
3044                                            <atom>N7</atom>
3045                                            <atom>C5</atom>
3046                                    </bond>
3047                                    <bond energyGroup="pr_n">
3048                                            <atom>C5</atom>
3049                                            <atom>C6</atom>
3050                                    </bond>
3051                                    <bond energyGroup="pr_n">
3052                                            <atom>C6</atom>
3053                                            <atom>N6</atom>
3054                                    </bond>
3055                                    <bond energyGroup="pr_n">
3056                                            <atom>C6</atom>
3057                                            <atom>N1</atom>
3058                                    </bond>
3059                                    <bond energyGroup="pr_n">
3060                                            <atom>N1</atom>
3061                                            <atom>C2</atom>
3062                                    </bond>
3063                                    <bond energyGroup="pr_n">
3064                                            <atom>C2</atom>
3065                                            <atom>N3</atom>
3066                                    </bond>
3067                                    <bond energyGroup="pr_n">
3068                                            <atom>N3</atom>
3069                                            <atom>C4</atom>
3070                                    </bond>
3071                                    <bond energyGroup="pr_n">
3072                                            <atom>C4</atom>
3073                                            <atom>C5</atom>
3074                                    </bond>
```

```
                                <bond energyGroup="pr_n">
                                        <atom>N9</atom>
                                        <atom>C4</atom>
                                </bond>
        <!--ADDITIONAL TO "A" FUNCTIONAL GROUP-->
                                <bond energyGroup="pr_n">
                                        <atom>C2</atom>
                                        <atom>S10</atom>
                                </bond>
                                <bond energyGroup="pr_n">
                                        <atom>S10</atom>
                                        <atom>C11</atom>
                                </bond>
                                <bond energyGroup="pr_n">
                                        <atom>N6</atom>
                                        <atom>C12</atom>
                                </bond>
                                <bond energyGroup="pr_n">
                                        <atom>C12</atom>
                                        <atom>C13</atom>
                                </bond>
                                <bond energyGroup="pr_n">
                                        <atom>C13</atom>
                                        <atom>C14</atom>
                                </bond>
                                <bond energyGroup="pr_n">
                                        <atom>C14</atom>
                                        <atom>C15</atom>
                                </bond>
                                <bond energyGroup="pr_n">
                                        <atom>C14</atom>
                                        <atom>C16</atom>
                                </bond>
                        </bonds>
```

Listing 7.3: Adding the bonds section to the residue structure

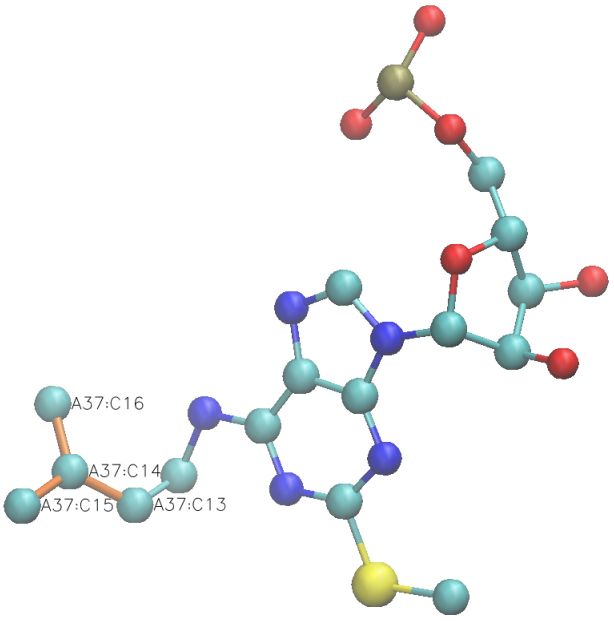
Note that the bonds are separated by the comments: `BACKBONE,FUNCTIONAL GROUP, ADDITIONAL BONDS TO 'RNA A' FUNCTIONAL GROUP`. The bond tag attribute `energyGroup` classifies the energy group the specific bond belongs to and helps to determine the dihedral strengths. The energy group `bb_n` refers to bonds that belong to the backbone group, and `pr_n` refers to the functional group. It is sometimes useful to use an existing residue as a reference if we know that the new residue only requires minor modifications. In our example, MIA is a modified RNA A molecule. All backbone and functional group bonds of RNA A can be added to MIA and we are left to determine the few other bonds created by the additional atoms: S10, C11-C16. The additions are added under the comment `ADDITIONAL BONDS TO 'RNA A' FUNCTIONAL GROUP` with energy group of `pr_n`.

7.3.4 List the improper dihedrals

Improper dihedrals cannot be dynamically calculated by the program using the bonds, and should be added separately. The tag `<improper></improper>` holds four atoms. The order of the four atoms here defines a specific improper dihedral within a biomolecule.

How to find an improper dihedral:

An improper dihedral is used to ensure proper geometry about a chiral center (i.e. prevent symmetry inversion due to an absent hydrogen atom). A proper dihedral would be defined by four sequential atoms connected by bonds (such as the dihedral C4*-C5*-O5*-P). An improper dihedral is defined by atoms that have a branched bonded geometry. Those angles need to be identified using the chemical structure of the molecule. For example we consider the four carbon atoms: C13,C14,C15,C16 and their bonds as defined above (highlighted in orange).



Finally, we add all of the improper dihedrals to our residue. In our example there is only one additional improper dihedral described above.

Add the improper dihedrals section to the residue structure:

```
3109                    <impropers>
```

```
3110                                    <improper>
3111                                            <atom>C3*</atom>
3112                                            <atom>C4*</atom>
3113                                            <atom>C5*</atom>
3114                                            <atom>O4*</atom>
3115                                    </improper>
3116                                    <improper>
3117                                            <atom>O3*</atom>
3118                                            <atom>C3*</atom>
3119                                            <atom>C4*</atom>
3120                                            <atom>C2*</atom>
3121                                    </improper>
3122                                    <improper>
3123                                            <atom>O2*</atom>
3124                                            <atom>C2*</atom>
3125                                            <atom>C1*</atom>
3126                                            <atom>C3*</atom>
3127                                    </improper>
3128                                    <improper>
3129                                            <atom>C2*</atom>
3130                                            <atom>C1*</atom>
3131                                            <atom>O4*</atom>
3132                                            <atom>N9</atom>
3133                                    </improper>
3134          <!--ADDITIONAL IMPROPER DIHEDRAL TO "RNA A" -->
3135                                    <improper>
3136                                            <atom>C13</atom>
3137                                            <atom>C14</atom>
3138                                            <atom>C15</atom>
3139                                            <atom>C16</atom>
3140                                    </improper>
3141                            </impropers>
3142                </residue>
```

LISTING 7.4: Adding the improper dihedrals section to the residue structure

7.4 Step 4 - Define a non-bonded group in the .nb file

Adding a new atom of a different mass requires a creation of a new non-bonded parameter definition. In our example we chose the atom sulfur to have a mass that is twice the mass of carbon. A new `<nonbond>` tag is added and it encapsulates the new non-bond group information such as mass charge and other non-bonded terms. The mass is doubled in the `mass` entry. The *nbType* includes the previously defined group name `NB_2` that is consistent with the .bif file.

Adding a new non-bonded type in the .nb fie:

```
<?xml version='1.0'?>
<nb>
 <!-- DEFAULTS -->
 <defaults gen-pairs="0"/>
 <!-- GENERAL NONBONDS -->
  <nonbond mass="2.00"  charge="0.000" ptype="A"  c6="0.0"  c12="5.96046e-9">
    <nbType>NB_2</nbType>
 </nonbond>
 <nonbond mass="1.00"  charge="0.000" ptype="A"  c6="0.0"  c12="5.96046e-9">
    <nbType>NB_1</nbType>
 </nonbond>
<!-- CONTACTS -->
<contact func="contact_1(6,12,?,energynorm)" contactGroup="c">
        <pairType>*</pairType>
        <pairType>*</pairType>
</contact>
</nb>
```

Listing 7.5: Defining a new non-bonded type in the .nb file

Chapter 8

Additional supported interaction types

As described above, it is the aim of SMOG 2 that the user will be able to extend the models in a wide range of ways. Accordingly, it is not possible that we provide descriptions of every possible variation that one may explore. However, here, we make an effort to provide some examples of how to implement specific features that we think may be frequently of interest.

8.1 Adding non-standard bonds between specific atoms

There are often cases where the covalent geometry of the system can not be determined by a general set of rules. For example, disulfide bonds may be formed, or broken, depending on the oxidation state. As another example, sugar structures often have branching patterns that do not form linear chains. For these types of chemical bonds, we provide the `BOND` option in the PDB file. If you would like to add a chemical bond, then add `BOND` lines immediately after the END line in the PDB file. This will add a bond, and all associated bond angles and dihedrals. Also, it will exclude contacts between any two atoms that are interacting through a bonded interaction (bond, angle, dihedral) that is defined by a BOND interaction.

prior to SMOG v2.4.5: The formatting is the following:

```
BOND ChainIndex1 AtomIndex1 ChainIndex2 AtomIndex2 energygroup
```

ChainIndex1 and AtomIndex1 indicate the first atom involved in the bond. ChainIndex1 is the index of the chain in which the atom exists, starting at 1. AtomIndex1 is the

number of the atom, *as it appears in the PDB file*. Note, ChainIndex1 is not necessarily the chain ID. ChainIndex2 and AtomIndex2 indicate the second atom involved in the bond. energygroup indicates the properties of any dihedral angles that have the new bond as a middle bond (See Chapter 6 for discussion on energy groups). For example, the following line:

```
BOND 1 51 4 100 r_p
```

would add a chemical bond between the atom numbered 51 in the first chain and the atom numbered 100 in the 4th chain. The bond properties would be determined based on the .b file settings, and the energy group of any associated dihedrals would be r_p. When SMOG 2 runs, it will write information to the screen as the `BOND` lines are detected. It will also write out information about what it interpreted the lines to mean, so you can verify that the intended bonds are added.

v2.4.5, and later: Support was added for BOND definitions with coarse-grained models. The formatting when using the -AA flag is the same as before:

```
BOND ChainIndex1 AtomIndex1 ChainIndex2 AtomIndex2 energygroup
```

However, if using the -CA or -tCG options, then the format is:

```
BOND ChainIndex1 ResidueIndex1 ChainIndex2 ResidueIndex2 energygroup
```

Note that all bond angles and dihedrals that may be defined using the newly-added BOND will be automatically generated. However, in v2.4.4 and earlier, only dihedrals that had the added bond as the central bond were added.

8.2 Including atom-type-specific parameters in a SMOG model

One of the goals of SMOG 2 is to allow one to "mix and match" elements of SMOG models and more highly-detailed energetic models (e.g. AMBER or CHARMM). For this, we have introduced the ability to define bonds, angles and dihedrals "by type". For example, if you would like all bonds between atoms of bType=B_1 and bType=B_2 to be given AMBER parameters, you could include the AMBER definition in the "extras" file (or any file with the suffix "extras" within the template directory) and then define the bond of type `bond_bytype` in the .b file. In this example, the .b file would have

```
<bond func="bond_bytype()">
<bType>B_1</bType>
```

```
<bType>B_2</bType>
</bond>
```

while the extras file would have

```
bondtypes < C OS 1 0.1323 10000
```

Notice in this example that the atom names in the extras file do not need to correspond to the values used for bTypes. This is intentional. In SMOG 2, each atom is assigned a bType, nbType and pairType. When determining which interaction to assign to a bond, the bType is read. However, when using bond_bytype, the type considered by Gromacs will be the atom type (which is the nbType in SMOG 2). So, in this example, if a residue were to have a C-O bond, and the C atom were bType=B_1 and nbType=C, while the O atoms were bType=B_2 and nbType=OS, then the bondtype parameters for C-OS would be used in the top file. When these templates are used to generate a top file, it would only list the atom numbers in the bonds section, while the parameters would appear under bondtypes.[1]

The extras file may be used to add content to any supported directive. However, nonbond_params, bondtypes, angletypes and dihedraltypes lines are only written if the atom types for each definition are used in a given SMOG model and simulated system. This ensures that the top file only contains information that is required for the given system.

For complete examples where AMBER terms have been added to SMOG models, see the SMOG 2 Force Field Repository.

8.3 Including perfectly free angles, dihedrals and contacts

In order to allow specific combinations of atom types to be given no interaction, the "free" interaction functions (`angle_free`, `dihedral_free`, and `contact_free`) are supported. They can be used as function types in the .b or .nb file. As an example, suppose you added a FRET dye as a new residue into the .bif:

```
<residue name="A594" residueType="ligand">
<atoms>
<atom bType="C" nbType="DYE" pairType="free">C1</atom>
<atom bType="C" nbType="DYE" pairType="free">C2</atom>
<atom bType="C" nbType="DYE" pairType="free">C3</atom>
...
```

[1] The format may vary when using the -OpenSMOG option

```
</atoms>
</residue>
```

To ensure that no native contacts would be defined with the FRET dye, the contact_free function must be used in the .nb file:

```
<contact func="contact_free()" contactGroup="c">
    <pairType>free</pairType>
    <pairType>*</pairType>
</contact>
```

Further, in order to ensure that exclusions are also not added for atoms that were identified as "in contact" (according to whichever contact protocol you have selected, such as Shadow, or cut-off), then you need to make sure that the function is declared in the .sif file, where the exclusions are explicitly disabled by setting the "exclusions" tag to zero:

```
<functions>
...
...
        <function name="contact_free" directive="pairs" exclusions="0"/>
</functions>
```

Together, this example would ensure that an atom of pairType `free` that is interacting with any other type (`*`) will be given a function `contact_free`, which adds nothing to the top file. As a note: matching `angle_free` also nullifies any dihedrals containing the angle. Using free functions will not change the normalizations for contact and dihedral interaction strengths, i.e. the contacts and dihedrals not written to the .top are still counted in the normalization sums. If you disable normalizations, then this will not be an issue.

8.4 Adding electrostatics

While not part of the default structure-based models distributed with SMOG 2, there are often times where it is desirable to add some degree of electrostatic interactions, or one would like to use interactions that are not of the 6-12 form.

There are two ways to introduce charges in your system. First, you can define the charge for a *nbType* in the .nb file, as shown in Listing 6.10. This will tell SMOG to assign a

charge to every atom of the specific *nbType*. The second way to add charges would be to define charges for specific atoms within a residue type, in the .bif file, as shown in Listing 6.1.

By default, Gromacs will treat electrostatic interactions as purely Coulombic. If you would like to use a screened electrostatic interaction (i.e. Debye-Hückel), then you need to supply a table to Gromacs, or use OpenSMOG/OpenMM to perform your simulations. If using Gromacs, the tool `smog_tablegen` will generate a screened-electrostatic look-up table, as described by Givaty and Levy[15].

Appendix A

Energetic Description of the Distributed Models

A.1 The All-Atom model

This model is selected with the `-AA` flag. All non-hydrogen atoms are explicitly represented, and the provided structure (i.e. the input PDB structure) is defined as the global potential energy minimum. Here, we provide a complete description of the default all-atom structure-based model energy function, which is defined by the template SBM_AA. All calculations employ reduced units (see A.4). Each atom is represented as a single bead of unit mass, and the charge of each atom is set to zero. Covalent geometry is maintained through harmonic interactions that ensure the bond lengths, bond angles, improper dihedral angles and planar dihedral angles remain about the values found in input structure. Non-bonded atom pairs that are in contact in the provided structure between residues i and j, where $i > j + 3$ for proteins and $i \neq j$ for RNA, are given an attractive 6-12 potential. The minimum of each 6-12 interaction is set to the distance of that atom pair in the provided structure. All non-native interactions between atoms that are not in contact in the native structure are repulsive. Contacts were defined according to the Shadow algorithm (See Chapter 5.8, with an all-atom cutoff distance of 6 Å and a shadowing radius of 1 Å. The functional form of the potential is,

$$
\begin{aligned}
V = & \sum_{\text{bonds}} \frac{\epsilon_{\text{r}}}{2}(r_i - r_{i,0})^2 + \sum_{\text{angles}} \frac{\epsilon_\theta}{2}(\theta_i - \theta_{i,0})^2 + \\
& \textstyle\sum_{\text{impropers}} \frac{\epsilon_{\chi_{\text{imp}}}}{2}(\chi_i - \chi_{i,0})^2 + \sum_{\text{planar}} \frac{\epsilon_{\chi_{\text{planar}}}}{2}(\chi_i - \chi_{i,0})^2 \\
& + \sum_{\text{backbone}} \epsilon_{\text{BB}} F_D(\phi_i - \phi_{i,0}) + \sum_{\text{sidechains}} \epsilon_{\text{SC}} F_{\text{D}}(\phi_i - \phi_{i,0}) \\
& + \sum_{\text{contacts}} \epsilon_{\text{C}} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \sum_{\text{non-contacts}} \epsilon_{\text{NC}} \left(\frac{\sigma_{\text{NC}}}{r_{ij}} \right)^{12}
\end{aligned}
\tag{A.1}
$$

where,

$$
F_{\text{D}}(\phi) = [1 - \cos(\phi)] + \frac{1}{2}[1 - \cos(3\phi)] \tag{A.2}
$$

When using SMOG 2, all values may be adjusted by the user, such as defining stabilizing non-native interactions and including non-specific dihedral angles. However, for the default model $r_{i,0}$, $\theta_{i,0}$, $\chi_{i,0}$, $\phi_{i,0}$ and σ_{ij} are given the values defined by the provided structure. For the default model, the parameters are set to the following values:

$\epsilon_{\text{r}} = 100\epsilon/\text{Å}^2$, $\epsilon_\theta = 80\epsilon/\text{rad}^2$, $\epsilon_{\chi_{imp}} = 10\epsilon/\text{rad}^2$, $\epsilon_{\chi_{planar}} = 40\epsilon/\text{rad}^2$, $\epsilon_{\text{NC}} = 0.1\epsilon$, $\sigma_{\text{NC}} = 2.5\text{Å}$, $\epsilon = 1$.

Note that, relative to the original implementation of this model, ϵ_r is decreased by a factor of two, ϵ_θ is increased and ϵ_{NC} is increased from 0.01ϵ to 0.1ϵ. Also, ω dihedrals are given the strength $\epsilon_{\chi_{\text{imp}}}$. As discussed in the SMOG 2 manuscript [16], this allows for a timestep of 0.002 to be utilized, which is larger than the originally-implemented 0.0005. ϵ is the reduced energy unit (see A.4) and has the value of 1 in the .top file which is equivalent to 1 kJ/mol if interpreted in Gromacs units. When assigning dihedral interaction weights (ϵ_{BB} and ϵ_{SC}), dihedrals are first grouped if they have a common middle bond. For example, in a protein backbone, there are up to four dihedral angles that may be defined that have the $C - C_\alpha$ bond as the middle bond. Each dihedral group is given a summed weight of ϵ_{BB}, or ϵ_{SC}. The ratio $R_{\text{BB/SC}} = \frac{\epsilon_{\text{BB}}}{\epsilon_{\text{SC}}}$ is set to 1 for nucleic acid dihedral angles and 2 for protein dihedral angles. ϵ_{BB} for protein and nucleic acids are equal. Dihedral strengths and contact strengths are scaled such that:

$$
R_{\text{C/D}} = \frac{\sum \epsilon_{\text{C}}}{\sum \epsilon_{\text{BB}} + \sum \epsilon_{\text{SC}}} = 2
$$

$$
\sum \epsilon_{\text{C}} + \sum \epsilon_{\text{BB}} + \sum \epsilon_{\text{SC}} = N\epsilon
$$

The sums are over all dihedral angles in the system, and N is the number of atoms in the system.

Note: The rescaling of dihedrals based on common middle bonds may not always desired. To explicitly indicate whether this feature should be enabled, add the following child element within the settings element in the .sif file

```
1  <dihedralNormalization dihedralCounting="0" />
```

0 indicates that counting should be turned off. 1 (default) indicates that counting should be performed.

A.2 The C_α model

This model is selected with the `-CA` flag. The C_α model coarse-grains the protein as single bead of unit mass per residue located at the position of the α carbon. $\vec{\mathbf{x_0}}$ denotes the coordinates of the native state and any subscript 0 signifies a value taken from the native state. The potential is given by

$$\begin{aligned} V_{\mathrm{C}\alpha}(\vec{\mathbf{x}},\vec{\mathbf{x_0}}) &= \sum_{\text{bonds}} \frac{\epsilon_{\mathrm{r}}}{2}(r-r_0)^2 + \sum_{\text{angles}} \frac{\epsilon_\theta}{2}(\theta-\theta_0)^2 + \sum_{\text{dihedrals}} \epsilon_{\mathrm{D}} F_{\mathrm{D}}(\phi-\phi_0) \\ &+ \sum_{\text{contacts}} \epsilon_{\mathrm{C}} \left[5\left(\frac{\sigma_{ij}}{r}\right)^{12} - 6\left(\frac{\sigma_{ij}}{r}\right)^{10} \right] + \sum_{\text{non-contacts}} \epsilon_{\mathrm{NC}} \left(\frac{\sigma_{\mathrm{NC}}}{r_{ij}}\right)^{12} \end{aligned} \tag{A.3}$$

where the dihedral potential F_{D} is,

$$F_{\mathrm{D}}(\phi) = [1-\cos(\phi)] + \frac{1}{2}[1-\cos(3\phi)]. \tag{A.4}$$

The coordinates $\vec{\mathbf{x}}$ describe a configuration of the α-carbons, with the bond lengths to nearest neighbors r, three-body angles θ, four-body dihedrals ϕ, and distance between atoms i and j given by r_{ij}. Protein contacts that are separated by less than 3 residues are neglected. Excluded volume is maintained by a hard wall interaction giving the residues an apparent radius of $\sigma_{\mathrm{NC}} = 4$ Å. The native bias is provided by using the parameters from the native state $\vec{\mathbf{x_0}}$. Setting the energy scale $\epsilon \equiv 1$, the coefficients are given the homogeneous values: $\epsilon_r = 200\epsilon/\text{Å}^2$, $\epsilon_\theta = 40\epsilon/\text{rad}^2$, $\epsilon_{\mathrm{D}} = \epsilon_{\mathrm{C}} = \epsilon_{\mathrm{NC}} = \epsilon$.

A.3 Gaussian contact potential (+gaussian templates)

Gaussian-shaped contact potentials (Fig. A.1) are added to an unofficial version of Gromacs that is available at smog-server.org, natively in OpenSMOG, and also in NAMD (see section A.3.4). These potentials are used when one desires control over either the shape of the excluded volume or the width of the attractive potential. They are also useful if a single contact requires multiple minima. In-depth characterization of the

Gaussian potentials with all-atom structure-based models using SMOG can be found in [13] (templates/SBM_AA+gaussian). They are explored in the context of a multi-basin C_α model here [17].

A.3.1 templates/SBM_AA+gaussian

This template can be selected using the `-AAgaussian` option. Selecting this template changes the contact potential to `ftype = 6`, ϵ_C=?, r_0=?, σ=$\sqrt{r_0^2/(50 \ln 2)}$, and enforces r_{NC}=0.21 nm. $r_{NC}^{12} = \epsilon_{NC} \times \sigma_{NC}^{12}$, *i.e.* the non-native excluded volume term, with ϵ_{NC} = 0.1 and σ_{NC} = 0.25 nm. The rather complex definition of the width of the Gaussian well σ is designed to model the variable width of the LJ potential: $C_{LJ}(1.2r_0) \sim -1/2$ so σ is defined such that $G(1.2r_0) = -1/2$ giving $\sigma^2 = (r_0)^2/(50 \ln 2)$.

- Note: v2.0 and earlier had r_{NC}=0.17 nm, i.e. $\epsilon_{NC} = 0.01\epsilon$ and σ_{NC}^{12} = 0.25 nm.

A.3.2 templates/SBM_calpha+gaussian

This template can be selected using the `-CAgaussian` option. Selecting this template changes the contact potential to `ftype = 6`, ϵ_C=?ϵ, r_0=? nm, σ=0.05 nm, r_{NC}=0.4 nm.

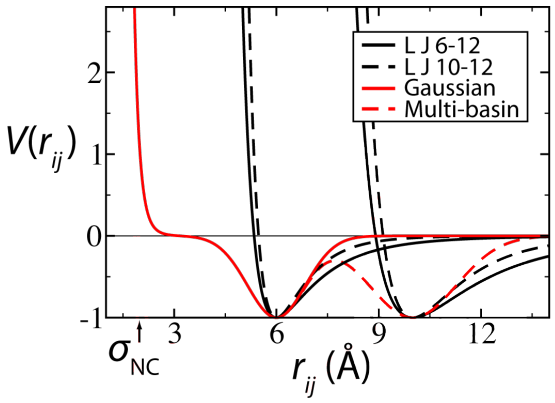


FIGURE A.1: Comparison of Lennard-Jones and Gaussian contact potentials. Black curves show LJ contact potentials with minima at 6 Å and 10 Å. The Gaussian contact potential shown in red has an excluded volume σ_{NC} (called r_{NC} elsewhere in the manual) that can be set independently of the location of the minimum. The dotted red line shows how the Gaussian contact would change as another minimum at 10 Å is added. Image adopted from [6].

A.3.3 Dual-basin Gaussian potential

Dual-basin interactions may be included when using OpenSMOG. For Gromacs, this is not yet implemented as a function in SMOG 2, but a description of ftype= 7, for use in an unofficial version of Gromacs, can be found on the SMOG webserver.

A.3.4 Downloading the source code extensions

A.3.4.1 OpenSMOG

OpenSMOG supports gaussian potentials without user-defined modifications.

A.3.4.2 Gromacs

The Gaussian contact shapes are not available in the standard Gromacs distributions. The necessary source code can be obtained at http://smog-server.org/extension/. This source distribution is compiled exactly as official Gromacs source distributions.

A.3.4.3 NAMD

Currently the "nightly build" version of NAMD contains the Gaussian potentials in the "Go potentials" section. More information can be found in the NAMD manual.

A.3.5 Including Gaussian potentials in the topology files

A.3.5.1 Gromacs

The Gaussian interaction is designated in the `[ pairs ]` section of the topology file.

- `ftype = 6`
  - $C_{ij}(r_{ij}) = -\epsilon_{\mathrm{C}} \left(\left(1 + \frac{1}{\epsilon_{\mathrm{C}}} \frac{r_{\mathrm{NC}}^{12}}{r_{ij}^{12}} \right) \left(1 - \exp\left[-\frac{(r_{ij} - r_{ij}^{0})^2}{2\sigma_{ij}^2} \right] \right) - 1 \right)$
  - $r_{\mathrm{NC}}^{12} = \epsilon_{\mathrm{NC}} \times \sigma_{\mathrm{NC}}^{12}$, where ϵ_{NC} and σ_{NC} come from the non-native excluded volume c12 term.
  - $\epsilon_{\mathrm{C}} \rightarrow$ depth of the attractive well
  - $r^{0} \rightarrow$ position of the minimum of the attractive well in nm
  - $\sigma \rightarrow$ width of the attractive well in nm

- $r_{\mathrm{NC}}^{12} \rightarrow$ position of the excluded volume hard wall in nm^{12}
- This form includes an excluded volume part, and therefore the pair ij should be included in `[ exclusions ]`. The multiplicative form anchors the minimum of the well at $(r_0, -\epsilon_{\mathrm{C}})$.

Note that `ftype = 5` and `ftype = 7` exist in the SBM extensions version, though there is no implementation in SMOG2 at the moment. They can be added by hand if desired.

A.3.5.2 NAMD

Currently the "nightly build" version of NAMD contains the Gaussian potentials in the "Go potentials" section. More information can be found in the NAMD manual.

A.4 Reduced units

SMOG topology files are written in reduced units, meaning that the masses, energies, and Boltzmann's constant are unitless and given a value of unity. (Note that lengths are written with units of nanometers for readability.) For example, this means that the all-atom model gives a mass of 1 to all the atoms with weights near 12 amu and the C-alpha model gives a mass of 1 for each coarse-grained residue bead. In such a scheme, the physically meaningful temperature is then near 1 (i.e. $\mathrm{k_B}T \approx 1$).

Independent of reduced units, interpreting the simulation temperatures used in SMOG models as real world temperatures can be challenging due to the mixing of effective energetic and entropic terms in a simplified potential. Gromacs adds an additional caveat to the expression of reduced temperatures because Gromacs uses a hard-coded value of Boltzmann's constant ($\mathrm{k_B}$) of 0.00831451 in Gromacs units of kJ/mol/K. Thus, a reduced temperature of 1 is expressed as 1/0.00831451 = 120.27 in a Gromacs .mdp file. You will find that proteins will typically unfold at reduced temperatures of 1-1.3 or Gromacs temperatures of 120-155. In OpenSMOG (v2.4.5 and later), reduced units are used consistently for temperature throughout all reporting. For a discussion of mixing externally known energetic terms with the effective energetics in SMOG (or equivalently, trying to estimate the reduced energy unit in physical units), see Wang et al [5].

One should also be careful when interpreting simulated time in SMOG models. Since these models (often) lack energetic roughness, the effective diffusion coefficients are elevated. As a results, numerical values of time are not directly meaningful. Instead, it is important to estimate the effective time of a simulated time unit (i.e. the number

reported as time). For an example, see Yang et al. [18], where diffusion coefficients in SMOG models and explicit-solvent simulations were compared for the ribosome. Since the effect of minimal roughness in SMOG models directly impacts diffusion, comparison of diffusion coefficients represents a sensible approach to estimating effective timescales. In contrast, comparing mean first-passage times for barrier-crossing events (e.g. a large-scale conformational change) can be extremely error-prone and lead to estimates that are off by orders of magnitude.

Appendix B

Installing Perl Modules using CPAN

B.1 Introduction

When installed Perl modules, it is often easiest to use CPAN, rather than manually installing the RPMs. Here we provide an introduction to CPAN to get you started. CPAN ("Comprehensive Perl Archive Network") is a collection of over 100,000 Perl modules ready to be installed. It is a great tool for easily adding Perl modules or updating your version of Perl.

B.2 Installing CPAN

Here we introduce installation for Linux and OSX. CPAN can be installed using the command line. If you have root privileges and are using the system Perl located at `/usr/bin/perl`, then it's easier to install via the package management system of your Linux or OSX distribution. You can explore CPAN options by typing "?".

RedHat/Fedora/CentOS Linux:

```
> sudo yum install perl-CPAN
```

Debian/Ubuntu Linux:

1. Install all dependent packages for CPAN:

```
> sudo apt-get install build-essential
```

2. Invoke the cpan command:

```
> cpan
```

3. Enter the commands below:

```
> make install
> install Bundle::CPAN
```

OSX on Mac:

Before you can use CPAN, you need to configure your Mac appropriately. CPAN uses some low-level tools to install modules, therefore tools like "Xcode" or the "Command Line Tools for Xcode package" are necessary. These are available at the Apple's Developer site. You should also have a terminal or shell application downloaded from the App store.

From the terminal application type the following and hit enter:

```
> cpan
```

This will take you through the installation process with more potential questions. You can enter "yes" for every question.

B.3 Upgrading your perl version using CPAN

You can upgrade your CPAN and perl version if it is not up to date. This step is recommended before installing any other modules.

1. In the CPAN application type:

```
> upgrade perl
```

2. If your perl or CPAN is not up to date, you can upgrade it:

```
> install CPAN
> reload cpan
```

```
cpan[2]> upgrade perl
CPAN: Storable loaded ok (v2.51)
Reading '/root/.cpan/Metadata'
  Database was generated on Fri, 18 Dec 2015 18:29:02 GMT
CPAN: LWP::UserAgent loaded ok (v6.06)
CPAN: Time::HiRes loaded ok (v1.9726)
Fetching with LWP:
http://www.perl.org/CPAN/authors/01mailrc.txt.gz
CPAN: YAML loaded ok (v1.13)
Reading '/root/.cpan/sources/authors/01mailrc.txt.gz'
CPAN: Compress::Zlib loaded ok (v2.067)
.....................................................................DONE
Fetching with LWP:
http://www.perl.org/CPAN/modules/02packages.details.txt.gz
Reading '/root/.cpan/sources/modules/02packages.details.txt.gz'
  Database was generated on Wed, 22 Jun 2016 18:53:39 GMT
.............
  New CPAN.pm version (v2.10) available.
  [Currently running version is v2.05]
  You might want to try
    install CPAN
    reload cpan
  to both upgrade CPAN.pm and run the new version without leaving
  the current session.


.........................................................DONE
Fetching with LWP:
http://www.perl.org/CPAN/modules/03modlist.data.gz
Reading '/root/.cpan/sources/modules/03modlist.data.gz'
DONE
Writing /root/.cpan/Metadata
All modules are up to date for perl

cpan[3]>
```

B.4 Example installation of a Perl module

Here we show an example of how to install the Perl module String::Util using CPAN.

1. Open your CPAN application as root (recommended) and type in:

```
> install String::Util
```

2. You should see some output messages in your terminal. If the installation is successful, you should see an output saying "Installation Complete" or "Build install OK".
This means you can go ahead and start using the module within your Perl scripts.

B.5 Troubleshooting tips

- Unsuccessful installation - Start by looking at the first error message that was output to the screen. Most likely there are missing libraries that are required before installing your module. It's recommended that you install those missing libraries

one by one, as instructed in the output messages. After installing the missing libraries, try the installation of the modules again.

- Changing the Perl version linked to CPAN - as mentioned earlier, by default CPAN works with the system Perl located at `/usr/bin/perl`. It can sometimes help to work with a different Perl version if your system Perl version is damaged and you cannot install modules properly. You can change which version CPAN sees by replacing a single line in the CPAN executable script:
  (i) Check the full path of CPAN executable by:
  `> which cpan`
  Usually the executable file is: `/usr/local/bin/cpan`
  (ii) Open (as root) the executable file, for example using vi:
  `> vi /usr/local/bin/cpan`

```
3  eval 'exec /usr/bin/perl  -S $0 ${1+"$@"}'
```

Listing B.1: CPAN default code

(iii) Change the following line (line n.3) to include the full path of the version you'd like to use:

```
3  eval 'exec /full/path/new/perl/version  -S $0 ${1+"$@"}'
```

Listing B.2: CPAN updated code

Save the changes in the file. The next time you use CPAN, all of the newly installed modules will be linked to the new perl version.

Appendix C

FAQs and Tips

Why is `grompp` so slow when using SMOG models?

In an effort to reduce the size of .tpr files, `grompp` tries to find parameters that are common (e.g. contacts with the same c6 and c12 parameters). In SMOG models, most contacts have unique parameters. This can lead to a very large number of unique parameters, which are then compared against all other parameters in the .top file. As a result, `grompp` executes a nested for loop that has increasing bounds (essentially an N^2 calculation). For small systems (less than $\sim 10,000$ atoms), this tends not to be a problem. However, for systems of 100,000 atoms or greater, `grompp` can easily require hours to complete. Since this bookkeeping feature of `grompp` does not appear to impact the performance of `mdrun` for SMOG models, it is desirable to disable this parameter comparison. To disable this check, you can make a one-line modification to the source code and rebuild Gromacs. For Gromacs v4.6.5, you will want to edit `src/kernel/convparm.c`. Specifically, go to line 465, which reads:

```
if (!bAppend)
```

and replace it with

```
if (0)
```

For Gromacs 5.1.4 (v 2021.2), the corresponding code can be found on line 542 (462) of

`src/gromacs/gmxpreprocess/convparm.c`

We have found that this modification results in a .tpr file that is 20-60% larger, though `grompp` can 100-1000 times faster.

I used `smog_adjustPDB` and when I run SMOG2 I get the error"FATAL ERROR: It appears that a residue in the PDB file does not contain all of the atoms defined in the .bif file.". What is wrong?

If you are NOT using the -legacy option: The only real explanation for this is that you are not using a mapping file that is consistent with the templates. If you are using default models, then you can use the default mapping file. However, if you are using non-standard templates, then you may need to use the -map option to specify which mapping file to use.

If you are using the -legacy option: Sometimes, this error arises because there were missing atoms in the PDB file (e.g. some atoms may have not been resolved in the crystallographic model). If you are convinced that all atoms are present, then there is another possibility. When you use smog_adjustPDB, it makes its best guesses for how residues should be named. For example, a 5'-terminal RNA residue (say G) will be renamed with a 5 added to the residue name (G5). In the default templates, G5 is defined as a residue that has a terminal phosphate group, however most PDB files are lacking these atoms. The appropriate name for a phosphate-less terminal G would be G0P (zero, not O). If this is the case, you can provide your own mapping file for smog_adjustPDB, or you can manually edit the residue names in the pdb file.

Appendix D

Acknowledgements

We are grateful to the NSF for long-term support to develop this software and provide it as a resource to the research community.

We would also like to thank everyone that has given feedback during the development of SMOG 2 (see THANKS file in the distribution).

We acknowledge Mark Howarth [19] for the protein structure alphabet used in the logo.

Bibliography

[1] C Clementi, H Nymeyer, and J N Onuchic. Topological and energetic factors: what determines the structural details of the transition state ensemble and "en-route" intermediates for protein folding? an investigation for small globular proteins. *J. Mol. Biol.*, 298(5):937–953, 2000.

[2] Paul C Whitford, Jeffrey K Noel, Shachi Gosavi, Alexander Schug, Kevin Y Sanbonmatsu, and José N Onuchic. An all-atom structure-based potential for proteins: bridging minimal models with all-atom empirical forcefields. *Proteins*, 75(2):430–441, 2009.

[3] PC Whitford, O Miyashita, Y Levy, and JN Onuchic. Conformational transitions of adenylate kinase: Switching by cracking. *J. Mol. Biol.*, 366(5):1661–1671, Jan 2007. doi: 10.1016/j.jmb.2006.11.085.

[4] Kien Nguyen and Paul C Whitford. Steric interactions lead to collective tilting motion in the ribosome during mRNA-tRNA translocation. *Nature Comm*, 7:10586–10586, 2016.

[5] Ailun Wang, Mariana Levi, Udayan Mohanty, and Paul C. Whitford. Diffuse Ions Coordinate Dynamics in a Ribonucleoprotein Assembly. *Journal of the American Chemical Society*, 144(21):9510–9522, 2022. ISSN 15205126. doi: 10.1021/jacs.2c04082.

[6] Jeffrey K Noel and José N Onuchic. The many faces of structure-based potentials: From protein folding landscapes to structural characterization of complex biomolecules. *Computational Modeling of Biological Systems, Springer US*, pages 31–54, 2012.

[7] J Bryngelson and P Wolynes. Intermediates and barrier crossing in a random energy model (with applications to protein folding). *J. Phys. Chem.*, 93:6902–6915, 1989.

[8] J D Bryngelson, J N Onuchic, N D Socci, and P G Wolynes. Funnels, pathways, and the energy landscape of protein folding: a synthesis. *Proteins*, 21(3):167–195, 1995.

[9] Sander Pronk, Szilárd Páll, Roland Schulz, Per Larsson, Pär Bjelkmar, Rossen Apostolov, Michael R Shirts, Jeremy C Smith, Peter M Kasson, David van der Spoel, Berk Hess, and Erik Lindahl. Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics*, 29(7):845–854, 2013.

[10] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kalé, and Klaus Schulten. Scalable molecular dynamics with namd. *J. Comput. Chem.*, 26 (16):1781–1802, 2005.

[11] Antonio B Oliveira, Vinicius G Contessoto, Asem Hassan, Sandra Byju, Ailun Wang, Yang Wang, Esteban Dodero-Rojas, Udayan Mohanty, Jeffrey K Noel, Jose N Onuchic, and Paul C Whitford. SMOG 2 and OpenSMOG: Expanding the limiting of structure-based models. *Protein Science*, 31:158–172, 2022.

[12] Jeffrey K Noel, Jorge Chahine, Vitor B P Leite, and Paul Charles Whitford. Capturing transition paths and transition states for conformational rearrangements in the ribosome. *Biophys J*, 107(12):2872–2881, December 2014.

[13] Jeffrey K Noel, Paul C Whitford, and José N Onuchic. The shadow map: a general contact definition for capturing the dynamics of biomolecular folding and function. *J. Phys. Chem. B*, 116(29):8692–8702, 2012.

[14] S Kumar, J Rosenberg, D Bouzida, and Swendsen. The weighted histogram analysis method for free-energy calculations on biomolecules. I. The method. *J. Comput. Chem.*, 13(8):1011, 1992.

[15] Ohad Givaty and Y Levy. Protein sliding along dna: dynamics and structural characterization. *J Mol Biol*, 385(4):1087–97, Jan 2009. doi: 10.1016/j.jmb.2008.11.016.

[16] Jeffrey K Noel, Mariana Levi, Mohit Raghunathan, Heiko Lammert, Ryan L Hayes, Jose N Onuchic, and Paul C Whitford. SMOG 2: A Versatile Software Package for Generating Structure-Based Models. *PLoS Comp Biol*, 12(3):e1004794, 2016.

[17] Heiko Lammert, Alexander Schug, and José N Onuchic. Robustness and generalization of structure-based models for protein folding and function. *Proteins*, 77(4): 881–891, 2009.

[18] Huan Yang, Prasad Bandarkar, Ransom Horne, Vitor B.P. Leite, Jorge Chahine, and Paul C. Whitford. Diffusion of tRNA inside the ribosome is position-dependent. *J. Chem. Phys*, 151(8), 2019. ISSN 00219606. doi: 10.1063/1.5113814.

[19] Mark Howarth. Say it with proteins: an alphabet of crystal structures. *Nat Struct Mol Biol*, 22:349, 2015.