GOMC
USER’S MANUAL
version 1.00

Distributed by the Potoff and Schwiebert Groups
© Wayne State University
# Table of Contents

Tutorial Overview ........................................................................................................... 3

Introduction (what is GOMC?) ..................................................................................... 3

How to get the software ............................................................................................... 3

Platform and Software Requirements ........................................................................... 4
  Supported Operating Systems ....................................................................................... 4
  Required Software Prequisites ..................................................................................... 5
  Highly Recommended Software Tools ........................................................................... 8
  Other Useful Software Tools ....................................................................................... 10

Compiling GOMC .......................................................................................................... 12
  Extracting the Code ..................................................................................................... 12
  Compiling the Code ..................................................................................................... 13
    GPU Code .................................................................................................................. 13
    Serial Code ............................................................................................................... 15

Input File Formats ......................................................................................................... 18
  PDB ............................................................................................................................... 18
  Parameter File(s) .......................................................................................................... 30
  Control File (in.dat) ..................................................................................................... 37

GOMC’s Output Files, Terminal Output ........................................................................ 56
  Console Output ............................................................................................................. 57
  PDB and PSF Files ....................................................................................................... 60
  Block Output Files ....................................................................................................... 60

Putting it all Together: Running a GOMC Simulation .................................................. 61
  Intermolecular Energy and Virial function .................................................................. 66

How to Get Help/Technical Support ............................................................................. 69
Tutorial Overview
This document will introduce a new user to how to download, compile and run the GOMC molecular simulation code. A working knowledge of statistical physics is recommended as a perquisite to understanding this tutorial.

To demonstrate the capabilities of the code, the user is guided through the process of downloading and compiling a GOMC executable. That executable is then used to perform saturated vapor liquid equilibria (VLE) studies on systems of pure isobutane (R600a), a branched alkane that is seeing increasing use as a refrigerant/propellant.

http://en.wikipedia.org/wiki/Isobutane

The Transferable Potentials for Phase Equilibria (TraPPE) united atom (UA) forcefield is used to describe the molecular geometry constraints and the intermolecular interactions.

Introduction (what is GOMC?)
Monte Carlo (MC) simulations are a kind of simulations that are driven by stochastic processes. The "GO" stands for GPU Optimized, as this code was intended to run optimally on modern graphics process hardware.

More specifically, this engine includes serial and GPU-optimized (multi-threaded) codes designed to run Markov chain Boltzmann sampling of chemical systems -- effectively sets of points defined by topological maps and interaction algorithms in a simulation box. From statistical mechanics we know this is one way to sample phase space and model chemical systems.

GOMC currently supports both canonical and Gibbs ensemble simulations. Support for GCMC and GEMC-NPT will be added shortly. GOMC uses widely used simulations file types (PDB, CHARMM-style parameter file, PSF). GOMC includes configurational bias algorithms for both linear and branched systems. Support for cyclic molecules and charged systems are being added.

How to get the software
Currently the latest public code builds, the project logo, manual, and other resources can be obtained via the website:
The code can be found under the download tab, just below and to the right of the logo. When new betas (or eventually release builds) are announced, they will replace the prior code under the downloads tab. An announcement will be posted on the front page to notify users.

Currently version control is handled through an internal SVN system maintained by the developers at Wayne State University. The posted builds are “frozen” versions of the code that have been validated for a number of systems and ensembles. Eventually the project will be hosted as a Git repository to allow for greater collaboration and a faster means of spotting new releases.

**Platform and Software Requirements**

**Supported Operating Systems**
GOMC officially supports Windows 7, 8, and most modern distributions of Linux (see the next section). This software may compile on recent versions of OS X, but that platform is not officially supported.
Required Software Prequisites

GOMC has some mild software requirements, which are widely available for Linux operating systems. Required software are:

1. C++03 Compliant Compiler
   a. Linux/OS X
      i. icc (Intel c++ compiler)
         Type...
         icc --version
         ...in a terminal. If gives a version number 4.4 or later, you’re all set. If it’s older than 4.4 (released in 2009), we recommend upgrading.

         In Linux, the Intel compiler will generally produce the fastest serial executables (when running on Intel Core processors).

      ii. g++ (GNU GCC)
          Type...
          g++ --version
          ...in a terminal. If gives a version number 4.4 or later, you’re all set. If it’s older than 4.4 (released in 2009), we recommend upgrading.

   b. Windows
      i. Visual Studio
         Microsoft’s Visual Studio 2010 or later is recommended.
         Check version:
         Help (top tab) → About Microsoft Visual Studio
2. `cmake` (if compiling on Linux)

To check if `cmake` is installed:
```
which cmake
```

To check version number:
```
cmake --version
```
Here’s an example from one of our systems:

nvcc/CUDA libs

The GPU builds of the code requires NVIDIA’s CUDA 6.0 or newer...

To check if nvcc is installed

```bash
which nvcc
```

To check version number:

```bash
nvcc --version
```

Here’s an example from one of our systems:

CUDA is viewed as an essential requirement, but is not used to compile the serial code, which can be compiled on systems without CUDA.

To download CUDA visit NVIDIA’s webpage:

CUDA is required to compile the GPU executable in both Windows and Linux. Please refer to CUDA Developer webpages to select an appropriate version for the desired platform.

To install CUDA in Linux, root/sudo privileges are generally required. In Windows, administrative access is required.

**Highly Recommended Software Tools**

**NOTE:**

These programs are used in this manual and are generally to be considered necessary for its examples.

**VMD**

VMD (Visual Molecular Dynamics) is a 3-D visualization and manipulation engine for molecular systems written in C-language. VMD is distributed and maintained by the University of Illinois at Urbana-Champaign. Its source and binaries are available free to download. It comes with a robust scripting engine capable of running python and tcl scripts.

More info can be found out here:

[http://www.ks.uiuc.edu/Research/vmd/](http://www.ks.uiuc.edu/Research/vmd/)

GOMC uses the same fundamental file types – PDB (coordinates) and PSF (topology) as VMD, although it uses some special tricks to obey certain rules of those file formats. One useful purpose of VMD is visualization of your systems.
A system of united atom isobutane molecules is seen above.

The most critical part of VMD, though is a tool called PSFGen. PSFGen uses a tcl or python script to generate a PDB and PSF file for a system of one or more molecules. It is perhaps the most convenient way to generate a compliant PSF file.

FIGURE: An overview of the PSFGen file generation process and its relationship to VMD/NAMD

To read more about PSFGen, please see:

Plugin homepage @ UIUC
http://www.ks.uiuc.edu/Research/vmd/plugins/psfgen/

“Generating a Protein Structure File (PSF)”, part of the NAMD Tutorial from UIUC
Packmol

Packmol is a molecule packing tool created by José Mario Martínez, a professor of mathematics at the State University of Campinas, Brazil. It is written in Fortran and is free to download. More information is available on its homepage:

http://www.ime.unicamp.br/~martinez/packmol/

To compile it a Fortran language compiler is needed, such as gfortran. Many Linux distributions no longer automatically come with Fortran compilers, so this may need to be installed.

Packmol allows a specified number of molecules to be packed at defined separating distances within a certain region of space. Packmol’s limitations include that it is unaware of topology – it treats each molecule or group of molecules it’s packing as a rigid set of points.

WARNING

Another more seriously limitation is that it is not aware of periodic boundary conditions (PBC). As a result, when using packmol to pack PDBs for GOMC, it is recommended to pack to a box 2 to 3 Angstroms smaller than the simulation box size. This prevents hard overlaps over the periodic boundary.

Other Useful Software Tools

Grace

Grace is a piece of graphing software written and maintained by the Weizmann Institute of Science's Plasma Laboratory (Rehovot, Israel). Mostly used in Linux, it can also be compiled in Windows, although the developers warn it may be missing some functionality.

In-depth information and the source can be found on the project page, here:

http://plasma-gate.weizmann.ac.il/Grace/
When compiled, Grace’s executable in Linux is typically named “xmgrace”. This tool allows the production of high quality, precise line and dot graphs. This makes it ideal for visualizing much of the thermodynamic data from the GOMC engine. Here is an example of the results of simulations of saturated VLE densities of linear alkanes produced with Grace:

Cygwin

Cygwin provides Microsoft Windows users with a Unix-like environment and command-line interface. It offers Windows-compatible ports of common Linux applications. It’s one option to assist in building and visualizing systems in Windows.

https://cygwin.com/

The software is free and open source, licensed under the GNU General Public License version 3. Its primary maintainers are Red Hat Inc. and NetApp. One of the most impressive abilities of Cygwin is its abil-
ity to launch a full Windows-compatible X-server Window, which allows convenient visualization of Linux app GUIs. It is compatible with the Grace graphing software. In practice, this package behaves most analogously to a Linux virtual machine in Windows.

**Compiling GOMC**

**Extracting the Code**

GOMC is distributed as a compressed folder which contains the source and build system. To compile the code after downloading it, the first step is to extract the compressed build folder.

In Windows the folder for the GPU code is compressed using a standard *.zip file format. To unzip simply use a utility like Peazip:

http://peazip.sourceforge.net/

Here’s an example of what the downloaded, code looks like when unzipping in Peazip...

In Linux the GPU and Serial codes are compressed using gzip and tar (*.tar.gz). To extract, simply move to the desire folder and type:

```
tar -xzvf <file name>.tar.gz
```
Compiling the Code

GPU Code

Compilation on Windows
Once the code is extracted, to compile it on Windows, you need to load the project into Visual Studio. To do that, open the extracted folder, and then double click on the solution file of the desired Visual Studio version (GOMCWin1_VS2010.sln or GOMCWin1_VS2012.sln). The current project has support for visual studio 2010 and 2012.

The default CUDA customization is CUDA 7.0. If you don’t have CUDA 7.0 installed on your machine, you need to change the .vcxproj file of the visual studio template you want to launch (example: template_vs2010.vcxproj). The file can be edited with a text editor. The change should be at the following tag (the file has two of them). You should change to the CUDA version you have installed on your machine:

```xml
<Import Project="$(VCTargetsPath)\BuildCustomizations\CUDA 7.0.props" />
```

You can later change the desired CUDA version from visual studio by selecting “Build Customizations” of the project (right click on the project in the solution explorer), then selecting the desired CUDA version.

To change the compute capability, you can go to the properties of the project (right click on the project in the solution explorer), then select “CUDA C/C++”, then device, then specify the desired compute capability in the “Code generation” text box.

After the solution is opened in Visual Studio, you can go to the “Build” menu and select “Build solution” to compile the code. You can compile either with release mode or debug mode by selecting the desired mode from the “Solution Configuration” drop box.

To run the project, simply click the run button or hit F5 on the keyboard.

Selecting Cell list optimization:
The GPU code has cell list support. To turn it on and off, you need to go to “calculateenergy.h”, and put “#define CELL_LIST” to turn the cell list support on.

Depending on how dense are your systems, the number of maximum atoms in the cell may need to be adjusted. As the GPU code has two cell lists (conventional and the micro cell list), you can go and increase the number of maximum atoms in the “system.h” file.

```c
// micro cell list
#define MAX_ATOMS_PER_CELL 18
// conv cell list
#define MaxParticleInCell 84
```

Selecting ensembles:
To compile the code with a desired ensemble, you need to go to “EnsemblePreprocessor.h”, and change the “ENSEMBLE” variable to one of the three supported ensembles (NVT, GCMC, GEMC).
**Selecting the build configuration (release vs debug) and 32 vs 64 platforms:**
To compile with release or debug, select the desired compile mode from the Build-> Configuration Manager menu. You can also select to build with 32 or 64 platforms from the same window.

To enable the debugging output in GOMC, you can go to the properties of the project (right click on the project in the solution explorer), then select C/C++, then Preprocessor, and then delete the “NDEBUG” preprocessor from the “Preprocessor definitions” text box.
Compilation on Linux

To compile the GPU code on Linux, just go into the directory of the project, and then type

```
make
```

the “makefile” file to choose different C compilers, select the desired compute capability, and to configure many more compilation flags.

The default compute capability is set to 3.0. To change the compute capability, go to the GENCODE_FLAGS option, and set it to one of the compute capability flags that are defined in the file.

```
# CUDA code generation flags
GENCODE_SM10 := -gencode arch=compute_10,code=sm_10
GENCODE_SM20 := -gencode arch=compute_20,code=sm_21
GENCODE_SM30 := -gencode arch=compute_30,code=sm_30
GENCODE_SM35 := -gencode arch=compute_35,code=sm_35
GENCODE_FLAGS := $(GENCODE_SM30)
```

To run the program, run the executable “GOMC.out”. To run this, the system’s LD_LIBRARY_PATH will need to be configured to support CUDA (more on this later).

You can refer to the previous sections to change ensembles, cell list support, debug output, and cell list size.

Serial Code

Compilation on Linux

In Linux the GPU code uses a simple makefile. Enter the directory and type

```
make all
```

... which will use the Makefile to compile a GPU-compatible executable called “GOMC.out”. To run this the system’s LD_LIBRARY_PATH will need to be configured to support CUDA (more on this later).

For the serial code – which uses cmake for compilation – simply go to the base directory and type:

```
./metamake.sh
```

This cmake script will create a directory named “bin”. Enter this directory:

```
cd bin
```

...and type:

```
make
```

Two executables – GOMC_Serial_GEMC (Gibbs ensemble) and GOMC_Serial_NVT (NVT ensemble) will be produced. By default the distribution compiles in release mode. To compile in debug mode (if you’re using the code as a developer) open the file “CMakeCache.txt”, while still in the “bin” folder. This file
contains information used by cmake to build the executables. To compile in debug mode, simply change the value after “CMAKE_BUILD_TYPE:STRING=” from “Release” to “Debug”...

```cmake
# This is the CMakeCache file.
# For build in directory: [REDACTED]/Serial_code/bin
# It was generated by CMake: /usr/bin/cmake
# You can edit this file to change values found and used by cmake.
# If you do not want to change any of the values, simply exit the editor.
# If you do want to change a value, simply edit, save, and exit the editor.
# The syntax for the file is as follows:
# KEY:TYPE=VALUE
# KEY is the name of a variable in the cache.
# TYPE is a hint to GUI's for the type of VALUE, DO NOT EDIT TYPE!
# VALUE is the current value for the KEY.

# EXTERNAL cache entries

###
//Path to a program.
CMAKE_AR:FILEPATH=/usr/bin/ar

//Choose the type of build, options are: None Debug Release RelWithDebInfo
// MinSizeRel
CMAKE_BUILD_TYPE:STRING=Release

//Enable/Disable color output during build.
CMAKE_COLOR_MAKEFILE:BOOL=ON

//CXX compiler.
CMAKE_CXX_COMPILER:FILEPATH=/usr/bin/c++
```
And retype the command:

```
make
```

The outputted executables should now be compiled with debugger symbols.

You can also swap the compiler by modifying the “CMAKE_CXX_COMPILER” variable. For more information refer to the Cmake documentation.
Input File Formats
A typical GOMC GEMC directory has the following:

- GOMC executable
- in.dat (proprietary control file)
- One (NVT ensemble) or two PDB files (Gibbs ensemble)
- One (NVT ensemble) or two PSF files (Gibbs ensemble)
- A CHARMM-style parameter file
- An exotic-style parameter file (may be empty, but must be present)

PDB
The PDB file stores coordinates for the simulation. The file format is widely adopted...

- Protein Databank (PDB) Files (plural: PDB files)
- Open format, well-documented
- Fixed-width format (hence white space is significant)
- Up to 13.5m page views a month; up to 55.8m FTP requests per month
- Used by NAMD, GROMACS, CHARMM, ACEMD, Amber

An overview of the PDB standard can be found here:

http://www.wwpdb.org/docs.html

The advantage of PDB files are their ubiquity and thorough documentation. Disadvantages include limited fixed point floating precision for coordinates, unused space, and proprietary implementations creating inconsistencies.

One PDB file is required per box, so for NVT ensemble simulations one file is expected, for Gibbs ensemble two files are required.

GOMC recognizes the following keywords in PDB files:

- REMARK
- CRYST1
- ATOM
- END

Currently REMARK is ignored (formerly it was used to store proprietary information in frames, e.g. step number, etc.). Note, packmol typically leaves the following remark:

REMARK original generated coordinate pdb file

...at the top of the file. Note this is another example of an inconsistency with the spec. As of the PDB v3.30 specification the REMARK entry contains an identifying integer, which is supposed to occupy lines
A file generated by packmol has “ori” in this position. Hence you may see future codes that are incompatible with this legacy kind of remarks.

Note also that the spaces 7 and 11 are not reserved; hence they may be used in proprietary specifications.

**CRYST1** can be used to store the cell dimensions, which can also be put as a tag in the proprietary control file.

http://www.wwpdb.org/documentation/format33/sect8.html#CRYST1
Crystallographic and Coordinate Transformation Section

This section describes the geometry of the crystallographic experiment and the coordinate system transformations.

CRYST1

Overview

The CRYS1 record presents the unit cell parameters, space group, and Z value. If the structure was not determined by crystallographic means, CRYS1 simply provides the unitary values, with an appropriate REMARK.

Record Format

<table>
<thead>
<tr>
<th>COLUMNS</th>
<th>DATA TYPE</th>
<th>FIELD</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 6</td>
<td>Record name</td>
<td>&quot;CRYS1&quot;</td>
<td></td>
</tr>
<tr>
<td>7 - 15</td>
<td>Real(9.3)</td>
<td>a</td>
<td>a (Angstroms).</td>
</tr>
<tr>
<td>16 - 24</td>
<td>Real(9.3)</td>
<td>b</td>
<td>b (Angstroms).</td>
</tr>
<tr>
<td>25 - 33</td>
<td>Real(9.3)</td>
<td>c</td>
<td>c (Angstroms).</td>
</tr>
<tr>
<td>34 - 40</td>
<td>Real(7.2)</td>
<td>alpha</td>
<td>alpha (degrees).</td>
</tr>
<tr>
<td>41 - 47</td>
<td>Real(7.2)</td>
<td>beta</td>
<td>beta (degrees).</td>
</tr>
<tr>
<td>48 - 54</td>
<td>Real(7.2)</td>
<td>gamma</td>
<td>gamma (degrees).</td>
</tr>
<tr>
<td>56 - 66</td>
<td>ASCII</td>
<td>sGroup</td>
<td>Space group.</td>
</tr>
<tr>
<td>67 - 70</td>
<td>Integer</td>
<td>z</td>
<td>Z value.</td>
</tr>
</tbody>
</table>

Details

- If the entry describes a structure determined by a technique other than X-ray crystallography, CRYS1 contains a = b = c = 1.0, alpha = beta = gamma = 90 degrees, space group = P 1, and Z = 1.
- The Hermann-Mauguin space group symbol is given without parenthesis, e.g., P 43 21 2. Please note that the screw axis is described as a two digit number.
- The full International Table’s Hermann-Mauguin symbol is used, e.g., P 1 21 1 instead of P 21.
- For a rhombohedral space group in the hexagonal setting, the lattice type symbol used is H.
- The Z value is the number of polymeric chains in a unit cell. In the case of heteropolymers, Z is the number of occurrences of the most populous chain.

As an example, given two chains A and B, each with a different sequence, and the space group P 2 that has two equivalent points in the standard unit cell, the following table gives the correct Z value.

Here’s an example...

REMARK
1 2 3 4 5 6 7
8
REMARK
89012345678901234567890123456789012345678901234567890123456789012345678901234567
890
CRYS1 71.490 71.490 71.490 90.00 90.00 90.00 P 1
ATOM 1 C1 ISB A 1 24.378 36.667 45.645 0.00 0.00
ISB C

This file indicates a box size of 71.49 Angstroms per side. The ‘90.00’ entries, indicate the cell is cubic.

NOTE:
Non-cubic cells are not yet supported in this code.
The main entry in the PDB file are ATOM entries. The keyword “ATOM” is always followed by two spaces. An entry has a number of fields.

---

**Coordinate Section**

The Coordinate Section contains the collection of atomic coordinates as well as the MODEL and ENMDL records.

**ATOM**

**Overview**

The ATOM records present the atomic coordinates for standard amino acids and nucleotides. They also present the occupancy and temperature factor for each atom. Non-polymer chemical coordinates use the HETATM record type. The element symbol is always present on each ATOM record. Charge is optional.

Changes in ATOM/HETATM records result from the standardization atom and residue nomenclature. This nomenclature is described in the Chemical Component Dictionary (http://wwwpdb.org/pdb/pdb/data/nomencl).

**Record Format**

<table>
<thead>
<tr>
<th>COLUMNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA TYPE</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>1 - 6</td>
</tr>
<tr>
<td>7 - 11</td>
</tr>
<tr>
<td>12 - 16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18 - 20</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23 - 26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>31 - 38</td>
</tr>
<tr>
<td>39 - 46</td>
</tr>
<tr>
<td>47 - 54</td>
</tr>
<tr>
<td>55 - 60</td>
</tr>
<tr>
<td>61 - 66</td>
</tr>
<tr>
<td>77 - 78</td>
</tr>
<tr>
<td>79 - 80</td>
</tr>
</tbody>
</table>

**Details**

- ATOM records for proteins are listed from amino to carboxyl terminus.
- Nucleic acid residues are listed from the 5' to the 3' terminus.
- Alignment of one-letter atom name such as C starts at column 14, while two-letter atom name such as FE starts at column 13.
- Atom nomenclature begins with atom type.
- No ordering is specified for polysaccharides.
- Non-blank alphanumeric character is used for chain identifier.
- The list of ATOM records in a chain is terminated by a TER record.
- If more than one model is present in the entry, each model is delimited by MODEL and ENMDL records.
- ALT LOC is the place holder to indicate alternate conformation. The alternate conformation can be in the entire polymer chain or several residues or partial residue (several atoms within one residue). If an atom is provided in more than one position, then a non-blank alternate location indicator must be used for each of the atomic positions. Within a residue, all atoms that are associated with each other in a given conformation are assigned the same alternate position indicator.
- There are two ways of representing alternate conformation either at atom level or at residue level (see examples).
- For atoms that are in alternate sites indicated by the alternate site indicator, sorting of atoms in the ATOM/HETATM list uses the following general rules:
  - In the simple case that involves a few atoms or a few residues with alternate sites, the coordinates occur once after the other in the entry.
  - In the case of a large heterogen groups which are disordered, the atoms for each conformation are listed together.
- Alphabet letters are commonly used for insertion code. The insertion code is used when two residues have the same numbering. The combination of residue numbering and insertion code defines the unique residue.
- If the depositor provides the data, then the isotropic B value is given for the temperature factor.
- If there are neither isotropic B values from the depositor, nor anisotropic temperature factors in ANISOU, then the default value of 0.0 is used for the temperature factor.
- Columns 79 - 80 indicate any charge on the atom, e.g., 2+, 1-. In most cases, these are blank.
- For refinements with program REFMAC prior 5.5.0442 which use TLS refinement, the values of B may include only the TLS contribution to the isotropic temperature factor rather than the full isotropic value.
The key parameters are the coordinates x, y, and z. The precision in these is limited to eight whole decimal digits and three fractional decimal digits.

Other important entries are the residue name, atom name, and chainID. Numbering is important primarily in that it represents an inconvenience in packing/loading large systems. Looking at the previous example:

```
REMARK
1234567890123456789012345678901234567890123456789012345678901234567890
C1 24.378 36.667 45.645
```

The atom name is “C1” and residue name is “ISB”. The PSF file (next section) contains a lookup table of atoms. These contain the atom name from the PDB and the name of the atom kind in the parameter file it corresponds to. As multiple different atom names will all correspond to the same parameter, these can be viewed “atom aliases” of sorts. The chain letter (in this case ‘A’) is sometimes used when packing a number of PDBs into a single PDB file.

### A few important Notes/Warnings on Undocument PDB Format Conventions:

- While it is only explicitly stated in some other sections of the PDB file, the general convention observed by most codes is to right align, when padding with white space.
- Some codes (including PSFGen/VMD) use the 21 unused character to add a fourth letter to the residue (molecule name). This extension is currently support, but is unofficial and hence may change in the future.
- VMD requires a constant number of ATOMs in a multi-frame PDB (multiple records terminated by “END” in a single file). To compensate for this all atoms from all boxes in the system are written to the outputted PDBs from this code.
- For atoms not currently in a box, the coordinates are set to <0.00, 0.00, 0.00>
- The occupancy is commonly just set to “1.00” and is left unused by many codes. We recycle this legacy parameter by using it to denote the box a particle is in, in our outputted PDBs (box 0 -> occupancy=0.00 ; box 1 -> occupancy=1.00)
- As the x, y, and z coordinates are fixed point with only three digits of precision, the energy values you get when restarting may be mildly different, particularly for bonded interactions due to roundoff in the coordinates. This will eventually be remedied by the implementation of a full-precision trajectory (e.g. DCD) file.
- The “ISB” entry in columns 73-75 is not an official part of the PDB standard. This is a proprietary entry called “Segname”, which has been embraced by NAMD and some other codes.
Note, currently a box is limited to holding up to 9,999 residues, or 99,999 atom entries, if sticking to decimal integer numbering. Like NAMD/X-PLOR, an upcoming build of the code will add support for using hexadecimal in the molecule numbering if 9,999 residues (see: NAMD) and using the alphanumeric overflow defined in the X-PLOR spec.

This will allow for a maximum of:

- \(9,999 \text{ (standard uint)} + 999 \times 26 \text{ (overflow, alphanumeric)} \rightarrow 35,973 \text{ molecules (residue #s)}\)
- \(99,999 \text{ (standard uint)} + 999 \times 26 \text{ (overflow, alphanumeric)} \rightarrow 359,973 \text{ ATOM entries}\)

Past either of these and an “overflow” solution is hit, where it will print stars to the respective field. This file should still be readable, but is consider unsafe in terms of portability, as it will work for GOMC and NAMD, but may fail for other codes.

A frame in the PDB file is terminated with the keyword **END**.

With that overview of the format in mind, here’s how a PDB file is typically built.

First, a single molecule PDB is obtained. In this example, the QM software package Gaussian was used to draw the molecule, which was then edited by hand to adhere to the PDB spec properly. The end result is a PDB for a single molecule:

```
REMARK   1 File created by GaussView 5.0.8
ATOM      1 C1   ISB     1       0.911 -0.313   0.000   C
ATOM      2 C2   ISB     1       1.424 -1.765   0.000   C
ATOM      3 C3   ISB     1      -0.629 -0.313   0.000   C
ATOM      4 C4   ISB     1       1.424  0.413  -1.257   C
END
```

Next packings are calculated, to place the simulation in a region of vapor-liquid coexistence. There are a couple of ways to do this in Gibbs ensemble:

1. Pack both boxes to a single “middle” density, which is an average of the liquid and vapor densities.
2. Same as 1, but add a modest amount to axis of one box (e.g. 10-30 A). This technique can be handy in the constant pressure Gibbs ensemble.
3. Pack one box to the predicted liquid density and the other to the vapor density.
A good reference for getting the information needed to estimate packing is the NIST Web Book database of pure compounds:

http://webbook.nist.gov/chemistry/

After packing is determined, a basic pack can be performed with a packmol script. Here is one example:

tolerance 3.0
filetype pdb
output STEP2_ISB_packed_Box_0.pdb

structure isobutane.pdb
number 1000
inside_box 0.0 0.0 70.20 70.20 70.20
end structure

Packmol scripts are typically saved with the extension *.inp, so this might be named “pack_isobutane.inp”.

To run the script, we type:

./packmol < pack_isobutane.inp

A more efficient way than packing individual simulations by hand is to embed a packmol script in a larger script, which reads in a table of comma or space separated values for the packing box sizes, as calculated via a spreadsheet.

Here’s an example snippet from tcl:

```tcl
./packmol < pack_isobutane.inp
```
When packing large files, packmol will adjust for this by incrementing the chain letter. This will cause issues with PSFGen. The solution is to use “grep” in Linux (or a similar method in Windows) to separate the PDB into several separate PDB files, one per chain. Each collection of residues of a specific kind will typically end up in a file together using this methodology.

The PDB output packmol is fed into PSFGen, along with a topology file. PSFGen will output a final PDB, which will be nearly identical to the packmol generated file. The only observable difference is generally the addition of a “segname”, the aforementioned field VMD/NAMD insert into an unspecified set of columns in ATOM entries. The two PDB files generated by PSFGen (or one if an NVT ensemble simulation is being prepped) will serve as the coordinates files for a new simulation.

**PSF File**

The PSF file stores the topology, mass, charges, and atom identities of molecules in the system.

- Protein Structure File (PSF)
- Space-separated file
- Used by NAMD, CHARMM, X-PLOR

The PSF file is not as robustly documented as the PDB format, but a basic description of it can be found here:


The PSF file is generally laid out in a series of sections. At the top of each section there is typically a line with a numeric value. This value lists the number of entries in that section (lines can contain multiple entries; a dihedral, for example has two quadruplet entries of atom indices per line). Note, that outside the remarks and atom section this number is typically smaller than the number of lines by a factor of 2.
PSF files always start with the string “PSF” on their first line.

GOMC reuses PSF reading code from NAMD, hence it should have much of the same flexibility and limitations. By section, the segments of a PSF file are:

- **TITLE:** remarks on the file
- **BONDS:** the bonds (if applicable) in molecules
- **ANGLE:** the bonds (if applicable) in molecules
- **DIHEDRAL:** the bonds (if applicable) in molecules
- **IMPROPER:** the bonds (if applicable) in molecules
- (other sections such as cross terms)

The code currently skips the title section and reads in the bonds, angles, dihedrals and impropers.

**A few important Notes/Warnings:**

- The PSF file format is a highly redundant file format. For example it repeats identical topology of thousands of molecules of a common kind, in some cases. GOMC follows the same approach as NAMD, allowing this excess information externally and compiling it in the code.
- Other sections (e.g. cross terms) are unsupported or legacy, hence are ignored.
- Following the restrictions of VMD, the order of the PSF atoms must match the order in the PDB file.
- Improper entries are read and stored, but are not currently used. Support will eventually be added for this.

The PSF file is typically generated using PSFGen. It is convenient to make a script to do this. For example the script:

```bash
psfgen << ENDMOL
topology ./Top_Branched_Alkanes.inp
segment ISB {
    pdb ./STEP2_ISB_packed_BOX_0.pdb
    first none
    last none
}
coordpdb ./STEP2_ISB_packed_BOX_0.pdb ISB
writepsf ./STEP3_START_ISB_sys_BOX_0.psf
writepdb ./STEP3_START_ISB_sys_BOX_0.pdb
```

Typically one script is run per box to generate a finalized PDB/PSF for that box. The script requires one additional file, the NAMD-style topology file. While GOMC does not directly read or interact with this
file, it’s typically used to generate the PSF and hence is considered one of the integral file types. It will be briefly discussed in the following section.

As with the packmol script, it’s often useful to pack the PSFGen script in a tcl or python script. Here’s an example:

```tcl
#(20) Pack copies of base model in first box.
exec packmol <<
  tolerance 3.0
  filetype pdb
  output step2_$(compound_shorthand)_packed_box_0.pdb
  structure step2_$(compound_shorthand).pdb
  number $(num_per_box)
  inside box 1, 1, 1: $(pack_1). $(pack_1). $(pack_1).
end structure

#(21) Pack copies of base model in second box.
exec packmol <<
  tolerance 3.0
  filetype pdb
  output step2_$(compound_shorthand)_packed_box_1.pdb
  structure step2_$(compound_shorthand).pdb
  number $(num_per_box)
  inside box 1, 1, 1: $(pack_v). $(pack_v). $(pack_v).
end structure

#(22) Run tcl script to build the final PDB/PSF for box 0
exec psfgen <<
  topology ./Top_Branched_Alkanes.inp
  segment $(compound_shorthand) {
    pdb ./step2_$(compound_shorthand)_packed_box_0.pdb
    first none
    last none
  }
  coordpdb ./step2_$(compound_shorthand)_packed_box_0.pdb $(c)
  writepsf ./step3_START_$(compound_shorthand)_sys_BOX_0.psf
  writedpb ./step3_START_$(compound_shorthand)_sys_BOX_0.pdb

#(23) Run tcl script to build the final PDB/PSF for box 1
exec psfgen <<
  topology ./Top_Branched_Alkanes.inp
  segment $(compound_shorthand) {
    pdb ./step2_$(compound_shorthand)_packed_box_1.pdb
    first none
    last none
  }
  coordpdb ./step2_$(compound_shorthand)_packed_box_1.pdb $(c)
  writepsf ./step3_START_$(compound_shorthand)_sys_BOX_1.psf
  writedpb ./step3_START_$(compound_shorthand)_sys_BOX_1.pdb
```

Here’s a peek at how the generated PSF file looks for a packed isobutane system (abridged):

```
PSF

3 !NTITLE
REMARKS original generated structure x-plor psf file
REMARKS topology ./Top_Branched_Alkanes.inp
REMARKS segment ISB { first NONE; last NONE; auto angles dihedrals }
```
4000 !NATOM
1 ISB  1    ISB  C1   CH1    0.000000       13.0190           0
2 ISB  1    ISB  C2   CH3    0.000000       15.0350           0
3 ISB  1    ISB  C3   CH3    0.000000       15.0350           0
4 ISB  1    ISB  C4   CH3    0.000000
5 ISB  2    ISB  C1   CH1    0.000000       13.0190           0
6 ISB  2    ISB  C2   CH3    0.000000       15.0350           0
7 ISB  2    ISB  C3   CH3    0.000000       15.0350           0
8 ISB  2    ISB  C4   CH3    0.000000       15.0350           0
9 ISB  3    ISB  C1   CH1    0.000000       13.0190           0
10 ISB  3    ISB  C2   CH3    0.000000       15.0350           0
3997 ISB  1000 ISB  C1   CH1    0.000000       13.0190           0
3998 ISB  1000 ISB  C2   CH3    0.000000       15.0350           0
3999 ISB  1000 ISB  C3   CH3    0.000000       15.0350           0
4000 ISB  1000 ISB  C4   CH3    0.000000       15.0350           0
3000 !NBOND: bonds
1  2  1  3  4  5  6
5  7  8  9 10  9 11
9 12 13 14 15 16
17 18 19 20 21 22
21 23 24 25 26 27
25 28 29 30 31 32
33 34 35 36 37 38
......
3977 3978 3981 3982 3981 3983 3981 3984 3985 3986 3985 3987 3985 3988 3989 3989
3989 3991 3989 3992 3993 3994 3993 3993 3995
3993 3996 3997 3998 3997 3999 3997 4000
3000 !NTHETA: angles
2  1  4  2  1  3  3  1  4
5  6  8  6  5  7  7  5  8
10 9 12 10 9 11 11 9 12
14 13 16 14 13 15 15 13 16
18 17 20 18 17 19 19 17 20
22 21 24 22 21 23 23 21 24
26 25 28 26 25 27 27 25 28
......
3990 3989 3992 3990 3989 3991 3991 3989 3992
3994 3993 3996 3994 3993 3995 3995 3993 3996
3998 3997 4000 3998 3997 3999 3999 3997 4000
0 !NPHI: dihedrals
0 !NIMPH: impropers
0 !NDON: donors
0 !NACC: acceptors
0 !NNB
0  0  0  0  0  0  0  0  0
Topology File

The topology is a whitespace separated file format which contains a list of atoms (with their masses), plus a list of residue information (charges, composition, and topology). It is essentially a non-redundant lookup table equivalent to the PSF file.

This is followed by a series of residues, which tell PSFGen what’s bonded to what. Each residue is comprised of four key elements:

- A header beginning with the keyword `RESI` with the residue name and net charge
- A body with multiple `ATOM` entries (not to be confused with the PDB-style entries of the same name) which list the partial charge on the particle and what kind of atom each named atom in a specific molecule/residue is.
- A section of lines starting with the word `BOND` containing pairs of bonded atoms (typically up to 3 per line)
- A closing section with instructions for PSFGen.

Here’s an example of a residue definition for isobutane:

```
RESI ISB 0.00 ! isobutane - TraPPE
GROUP
ATOM C1 CH1 0.00 ! C3
ATOM C2 CH3 0.00 ! C2-C1
ATOM C3 CH3 0.00 ! C4
ATOM C4 CH3 0.00 !
BOND C1 C2 C1 C3 C1 C4
PATCHING FIRS NONE LAST NONE
```

Here’s a full parameter file prepared to pack a system of isobutane:

```
* Custom top file -- branched alkanes
*
 1 1
!
MASS 1 CH3 15.035 C !
MASS 3 CH1 13.019 C !
RESI ISB 0.00 ! isobutane - TraPPE
GROUP
ATOM C1 CH1 0.00 ! C3
ATOM C2 CH3 0.00 ! C2-C1
ATOM C3 CH3 0.00 ! C4
ATOM C4 CH3 0.00 !
BOND C1 C2 C1 C3 C1 C4
PATCHING FIRS NONE LAST NONE
```
Note the keyword **END** must be used to terminate this file and keywords related to the autogeneration process must be placed near the top of the file, after the **MASS** definitions.

More in-depth information can be found in the following links...

- “Topography Tutorial” (PDF, in-depth)
- “NAMD Tutorial: 4. Examining the Topology File”
- “Developing Topology and Parameter Files”
- “NAMD Tutorial: 25. Topology Files”

…courtesy of UIUC

**Parameter File(s)**

Currently GOMC uses a single parameter file. However the user must specify from two choices of the kind of parameter file:

- CHARMM (Chemistry at HARvard Molecular Mechanics) compatible parameter file
- "EXOTIC" parameter file

If the parameter file type is not specified or if the chosen file is missing an error will be thrown. Both forcefield file options are whitespace separated files, with sections preceded by a tag. When a known tag (representing a kind of molecular interaction in the model) is encountered, reading of that section of the forcefield begins. Comments (anything after a * or !) and whitespace are ignored. Reading concludes when the end of the file is reached or another section tag is encountered.

**CHARMM format parameter file**

CHARMM contains a widely used model used in Monte Carlo and molecular dynamics simulations for describing energies. It is intended to be compatible with other codes that use this format, such as NAMD.

For a general overview of the CHARMM forcefield, see:

http://www.charmmtutorial.org/index.php/The_Energy_Function
Here’s the basic CHARMm contributions:

\[
U_{\text{CHARMM}} = U_{\text{bonds}} + U_{\text{non-bonded}}
\]

where \( U_{\text{bonds}} \) consists of the following terms:

\[
U_{\text{bonds}} = U_{\text{bond}} + U_{\text{angle}} + U_{\text{dihedra}} + U_{\text{improper}} + U_{\text{CMAP}}
\]

with

\[
U_{\text{bond}} = \sum_{\text{bonds}} K_b (b - b^0)^2,
\]

\[
U_{\text{angle}} = \sum_{\text{angles}} K_\theta (\theta - \theta^0)^2,
\]

\[
U_{\text{dihedra}} = \sum_{\text{dihedrals}} K_\phi (1 + \cos (n\phi - \delta))
\]

\[
U_{\text{improper}} = \sum_{\text{impropeers}} K_\omega (\omega - \omega^0)^2, \text{ and}
\]

\[
U_{\text{CMAP}} = \sum_{\text{residues}} u_{\text{CMAP}} (\phi, \psi)
\]

\[U_{\text{non-bonded}} \text{ consists of two terms,}
\]

\[U_{\text{non-bonded}} = U_{\text{LJ}} + U_{\text{elec}}\]

with

\[U_{\text{LJ}} = \sum_{\text{nonb. pairs}} \epsilon_{ij} \left( \frac{r_{ij}^{12}}{r_{ij}^6} - 2 \frac{r_{ij}^{12}}{r_{ij}^6} \right),\]

\[U_{\text{elec}} = \sum_{\text{nonb. pairs}} \frac{q_i q_j}{r_{ij}}\]

\[\checkmark \quad \text{Fully supported}\]

\[\bigcirc \quad \text{Read, but not supported}\]

\[\times \quad \text{Not currently Supported or read}\]

As seen above, the following are recognized, read and used:

- **BONDS**
  - Quadratic expression describing bond stretching based on bond length \( b \) in Angstrom
  - Typically is ignored as bonds are rigid for Monte Carlo sims. To specify that it is to be ignored put a very large value i.e. "999999999999" for \( K_b \).
• **ANGLES**  
  o Describes the conformational behavior of an angle ($\theta$) between three atoms, one of which is shared branch point to the other two.

  ![Angle Diagram](Image Courtesy of Wikimedia Commons)

• **DIHEDRALS**  
  Describes crankshaft like rotation behavior about a central bond in a series of three consecutive bonds (rotation is given as $\varphi$)

  ![Dihedral Diagram](Image Courtesy of Wikimedia Commons)

• **NONBONDED**  
  This tag name only should only be used if CHARMM force field is being used. This section describes 12-6 (Lennard-Jones) nonbonded interactions. Nonbonded parameters is assigned by specifying atom type name followed by polarizabilities (will be ignored), minimum energy and (minimum radius)/2. In order to modify 1-4 interaction, a second polarizabilities (will be ignored), minimum energy and (minimum radius)/2 need to be defined otherwise same parameter will be considered for 1-4 interaction.
• **NBFIX**  This tag name only should only be used if CHARMM force field is being used. This section allows interaction between two pair of atoms be modified. This is done by specifying two atoms type name followed by minimum energy and minimum radius. In order to modify 1-4 interaction, a second minimum energy and minimum radius need to be defined otherwise same parameter will be considered for 1-4 interaction.

**NOTE:**

Please pay attention that in this section we define minimum radius, not (minimum radius)/2 as it is defined in NONBONDED section.

Currently supported sections of the CHARMM compliant file include **BONDS, ANGLES, DIHEDRALS, IMPROPER, NONBONDED, NBFIX.** Other sections such as **CMAP** are not currently read or supported.

**BONDS** (aka “bond stretching”) are one key sections of the CHARMM-compliant file. Units for the $K_b$ variable in this section are in kcal/mol (for $K_b$; the $b_0$ section (which represents the default bond length for that kind of pair) is measured in Angstroms.

**BONDS**

\[ V(bond) = K_b(b - b_0)^2 \]

\[ K_b: \text{kcal/mole/A}^2 \]

\[ b_0: \text{A} \]

\[ K_b (\text{kcal/mol}) = K_b (K) \times \text{Boltz. const.}; (9999999999 if no stretching) \]

<table>
<thead>
<tr>
<th>atom type</th>
<th>$K_b$</th>
<th>$b_0$</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH3 CH1</td>
<td>9999999999</td>
<td>1.540</td>
<td>Trappe 2</td>
</tr>
</tbody>
</table>
**NOTE:**
The Kb value may appear odd, but this is because a larger value corresponds to a more rigid bond. As Monte Carlo forcefields (e.g., TraPPE) typically treat molecules as rigid constructs, Kb is set to a large value -- 9999999999.

Next comes an **ANGLES** (bond bending) section. $\theta$ and $\theta_0$ are commonly measured in degrees and $K_{\theta}$ is measured in kcal/mol/K. These values are often given in the literature in Kelvin (K). To convert Kelvin to kcal/mol/K, multiply by the Boltzmann constant -- $k_b$, $0.0019872041$ kcal/mol.

Here is an example of what is necessary for isobutane:

```
! V(angle) = Ktheta(Theta - Theta0)**2
! V(Urey-Bradley) = Kub(S - S0)**2
! Ktheta: kcal/mole/rad**2
! Theta0: degrees
! Kub: kcal/mole/A**2 (Urey-Bradley)
! S0: A
! Ktheta (kcal/mol) = Ktheta (K) * Boltz. const.
! atom types         Ktheta       Theta0   Kub(?)  S0(?)
CH3  CH1  CH3         62.100125    112.00 ! TraPPE 2
```

Some CHARMM **ANGLES** section entries include Urey-Bradley potentials ($K_{ub}$, $b_{ub}$), in addition to the standard quadratic angle potential. The constants related to this potential function are currently read in, but the logic has not been added to calculate this potential function. Support for this potential function will be added in later versions of the code.

The final major bonded interactions section of the CHARMM compliant parameter file are the **DIHEDRALS**. Each dihedral is composed of a dihedral series of 1 or more terms. Often there are 4 to 6 terms in a dihedral. Angles (for the dihedrals’ deltas) are given in degrees.

(Since isobutane has no dihedral, here’s the parameters pertaining to 2,3-dimethylbutane.)

```
! V(dihedral) = Kchi(1 + cos(n(chi) - delta))
! Kchi: kcal/mole
! n: multiplicity
! delta: degrees
```
\[ K_{\text{chi}} \text{ (kcal/mol)} = K_{\text{chi}} \text{ (K)} \times \text{Boltz. const.} \]

<table>
<thead>
<tr>
<th>atom types</th>
<th>Kchi</th>
<th>n</th>
<th>delta</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X CH1 CH1 X</td>
<td>-0.498907</td>
<td>0</td>
<td>0.0</td>
<td>! TraPPE 2</td>
</tr>
<tr>
<td>X CH1 CH1 X</td>
<td>0.851974</td>
<td>1</td>
<td>0.0</td>
<td>! TraPPE 2</td>
</tr>
<tr>
<td>X CH1 CH1 X</td>
<td>-0.222269</td>
<td>2</td>
<td>180.0</td>
<td>! TraPPE 2</td>
</tr>
<tr>
<td>X CH1 CH1 X</td>
<td>0.876894</td>
<td>3</td>
<td>0.0</td>
<td>! TraPPE 2</td>
</tr>
</tbody>
</table>

**NOTE:**
The code allows the use of ‘X’ to indicate ambiguous positions on the ends. This is useful as this kind are often determined solely by the two middle atoms in the middle of the dihedral, according to the literature.

**IMPROPERS** – energy parameters used to describe out-of-plane rocking are currently read in, but unused. The section is often blank. If it becomes necessary, algorithms to calculate the improper energy will need to be added.

The next section of the CHARMM style parameter file is the **NONBONDED**. In order to use TraPPE this section of the CHARMM compliant file will be critical. Here’s an example with our isobutane potential models.

**NONBONDED**

\[
V(\text{Lennard-Jones}) = \epsilon_{i,j}[(R_{\text{min},i,j}/r_{i,j})^{12} - 2(R_{\text{min},i,j}/r_{i,j})^{6}]
\]

\( \epsilon_{i,j} \text{: kcal/mole, } \epsilon_{i,j} = \sqrt{\epsilon_i \epsilon_j} \)

\( R_{\text{min},i,j} = R_{\text{min}/2,i} + R_{\text{min}/2,j} \)

\( R_{\text{min}} = \sigma * (2^{(1/6)}) / 2 \); \( \epsilon_{i,j} \text{ (kcal/mol)} = \epsilon_i \text{ (K)} \times \text{Boltz. const.} \)

**NOTE:**
Beware, the \( R_{\text{min}} \neq \sigma \). \( R_{\text{min}} \) is the distance to the x-intercept (where interaction energy goes from being repulsive to positive). \( \sigma \) is the distance to the deepest part of the well, where the attraction is maximum. To convert to \( R_{\text{min}} \), simply multiply \( R_{\text{min}} \) by 0.56123102415, and flag it with a negative sign.

The last section of the CHARMM style parameter file is the **NBFIX**. Here’s an example if it is required to modify interaction between CH3 and CH1 atoms.

**NBFIX**

\[
V(\text{Lennard-Jones}) = \epsilon_{i,j}[(R_{\text{min},i,j}/r_{i,j})^{12} - 2(R_{\text{min},i,j}/r_{i,j})^{6}]
\]

\( \epsilon_{i,j} \text{: kcal/mole, } \epsilon_{i,j} = \sqrt{\epsilon_i \epsilon_j} \)

\( R_{\text{min},i,j} = R_{\text{min}/2,i} + R_{\text{min}/2,j} \)

\( R_{\text{min}} = \sigma * (2^{(1/6)}) / 2 \); \( \epsilon_{i,j} \text{ (kcal/mol)} = \epsilon_i \text{ (K)} \times \text{Boltz. const.} \)
"Exotic" parameter file

The exotic file is intended for use with nonstandard/specialty models of molecular interaction which are not included in CHARMM standard. Currently a single custom interaction is included:

- **NONBONDED_MIE** This section describes n-6 (Lennard-Jones) nonbonded interactions. The Lennard-Jones potential (12-6) is a subset of this potential. Nonbonded parameters is assigned by specifying atom type name followed by minimum energy, atom diameter and repulsion exponent. In order to modify 1-4 interaction, a second minimum energy, atom diameter and repulsion exponent need to be defined otherwise same parameters will be considered for 1-4 interaction.

- **NBFIX_FIE** This section allows n-6 (Lennard-Jones) interaction between tow pair of atoms be modified. This is done by specifying two atoms type name followed by minimum energy, atom diameter and repulsion exponent. In order to modify 1-4 interaction, a second minimum energy, atom diameter and repulsion exponent need to be defined otherwise same parameter will be considered for 1-4 interaction.

**NOTE:**

In EXOTIC force field the definition of atom diameter(σ) is same for both NONBONDED_MIE and NBFIX_FIE.

Otherwise the Exotic file reuses the same geometry section headings -- BONDS / ANGLES / DI-HEDRALS / etc. The only difference in these sections versus in the CHARMM format forcefield file is that the energies are in Kelvin (‘K’), the unit most commonly found for parameters in Monte Carlo chemical simulation literature. This precludes the need to convert to kcal/mol, the energy unit used in CHARMM.

The most frequently used section of the exotic files in the Mie potential section, NONBONDED_MIE.

Here are the parameters that are used to simulate alkanes.

```
NONBONDED_MIE
! V(mie) = 4*eps*((sig_ij/r_ij)^n-(sig_ij/r_ij)^6)
```

```
Here is an example if there is need to modify CH3 CH2 interaction.

\[
V_{\text{mie}} = 4*\varepsilon ((\sigma_{ij}/r_{ij})^n - (\sigma_{ij}/r_{ij})^6)
\]

Note the exotic file uses $\sigma$, not the $R_{\text{min}}$ used by CHARMM, although the units (Angstroms) are the same. The energy in the exotic file are expressed in Kelvin (K), as this is the standard convention in the literature.

**Control File (in.dat)**

The control file is GOMC’s proprietary input file. It contains key settings. The settings generally fall under three categories:

- **Input/Simulation Setup**
- **System Settings for During Run**
- **Output Settings**

**NOTE:** Control file is designed in order to recognize logic value such as yes/true/on or no/false/off.

**Input/Simulation Setup**

In this section input file names are listed. In addition if you want to restart your simulation or use integer seed for running your simulation you need to modify this section according to your purpose.

- **Restart**: Determines whether to restart, and if so what step to restart from.

  *(WARNING: Restarts are not currently supported)*

  o Value 1: <BOOLEAN> – true if restart, false otherwise

- **FirstStep**: Determines what step to restart from. If Restart was set to true, step number need to be specified, otherwise program will be terminated.

  o Value 1: <ULONG> – step to restart from

Example:

```
#..............................................................
```
# enable, step

```
# Restart   true
FirstStep  1000000
```

- **PRNG**: Dictates how to start the pseudo-random number generator (PRNG).
  - Value 1: <STRING>
    - **RANDOM**: Randomizes Mersenne Twister PRNG with random bits based on the system time.
      
      Example:
      ```
      # kind {RESTART, RANDOM, INTSEED}
      PRNG RANDOM
      ```
    - **RESTART**: Used for restarting a previous simulation, this option loads in the Mersenne Twister’s state from a saved file.
      
      *(WARNING: Restarts are not currently supported)*
      
      Example:
      ```
      # kind {RESTART, RANDOM, INTSEED}
      PRNG RESTART
      ```
    - **INTSEED**: This option “seeds” the Mersenne Twister PRNG with a standard integer. When the same integer is used, the generated PRNG stream should be the same every time, which is helpful in tracking down bugs.

- **Random_Seed**: Define seed number. If INTSEED was chosen, seed number need to be specified, otherwise program will be terminated.
  - Value 1: ULONG or UINT: If “INTSEED” command is used (see above example).
    
    Example:
    ```
    # kind {RESTART, RANDOM, INTSEED}
    PRNG INTSEED
    Random_Seed 5
    ```

- **ParaTypeCHARMM**: Sets forcefield type to CHARMM style.
  - Value 1: <BOOLEAN> – true if it is CHARMM forcefield, false if it is not.
    
    Example:
    ```
    # ParaTypeCHARMM
    ```
• **ParaTypeEXOTIC**: Sets forcefield type to EXOTIC style.
  
  o Value 1: <BOOLEAN> – true if it is charm forcefield, false if it is not.

  Example:
  ```
  # FORCE FIELD TYPE
  # FORCE FIELD TYPE
  ParaTypeCHARMM = true
  ParaTypeEXOTIC = Set
  ```

• **Parameters**: Provides the name of the parameter file to use for the simulation

  o Value 1: <STRING> – Sets the name of parameter file

  Example:
  ```
  # FORCE FIELD TYPE
  # FORCE FIELD TYPE
  ParaTypeCHARMM = yes
  Parameters = Par_TrapPPE_Alkanes.inp
  ```

• **Coordinates**: Defines the PDB filenames (coordinates) for each box in the system

  o Value 1: <INTEGER> – Sets box number (first box is box ‘0’)
  
  o Value 2: <STRING> – Sets pdb file name

  Note:
  NVT ensemble requires only one PDB file and GEMC/GCMC require two PDB files. If number of PDB files were not compatible to simulation type, program will be terminated.

  Example (NVT ensemble):
  ```
  # INPUT PDB FILES
  Coordinates 0 STEP3.START.ISB.sys.pdb
  ```

  Example (Gibbs ensemble or GC ensemble):
  ```
  # INPUT PDB FILES
  Coordinates 0 STEP3.START.ISB.sys_BOX_0.pdb
  ```
**Coordinates 1**  
**STEP3_START_ISB_sys_BOX_1.pdb**

- **Structures**: Defines the PSF filenames (structures) for each box in the system
  
  - Value 1: `<INTEGER>` – Sets box number (first box is box ‘0’)
  - Value 2: `<STRING>` – Sets PSF file name

  **Note:**

  NVT ensemble requires only one PSF file and GEMC/GCMC require two PSF files. If number of PSF files were not compatible to simulation type, program will be terminated.

  **Example (NVT ensemble):**
  
  `####################################
  # INPUT PSF FILES
  ####################################
  Structure 0  STEP3_START_ISB_sys.psf`

  **Example (Gibbs ensemble or GC ensemble):**
  
  `####################################
  # INPUT PSF FILES
  ####################################
  Structure 0  STEP3_START_ISB_sys_BOX_0.psf
  Structure 1  STEP3_START_ISB_sys_BOX_1.psf`

**System Settings for During Run Setup**

This section contains all the variables not involved with the output of data during the simulation or with the reading of input files at the start of the simulation. In other words, it contains settings related to the moves, the thermodynamic constants (based on choice of ensemble), and the length of the simulation.

Note that some tags or entries for tags are only used in certain ensembles (e.g. Gibbs ensemble). These cases are denoted with colored text.

- **GEMC**: *(FOR Gibb ensemble runs only)* Defines what type of Gibbs Ensemble simulation you want to run.

  If neglected in Gibbs ensemble, it simply defaults to constant volume (NVT) Gibbs ensemble

  - Value 1: `<STRING>` – allows you to pick between isovolumetric (“NVT”) and isobaric (“NPT”) Gibbs ensemble simulations
    - NVT: Run simulation w/ constant mole number, volume and temperature.

    **Example:**
    
    `####################################`
• **NPT**: Run simulation w/ constant mole number, pressure and temperature.

• **Pressure**: If NPT simulation was chosen, pressure need to be specified, otherwise program will be terminated.
  - Value 1: <DOUBLE> – Constant pressure in bars

Example:
```
# GEMC TYPE (DEFAULT IS NVT_GEMC)  
GEMC NVT
GEMC NPT
Pressure 5.76
```

• **Temperature**: Sets the temperature that system will run at.
  - Value 1: <DOUBLE> – constant temperature of simulation in Kelvin

• **Rcut**: Sets a specific radius that nonbonded interaction energy and force will be considered and calculated using defined potential function.
  - Value 1: <DOUBLE> – the distance to calculated the Lennard-Jones potential at

• **LRC**: Defines whether to use long range corrections.
  - Value 1: <BOOLEAN> – True to consider long range correction. In case of using SHIFT or SWITCH potential function, LRC will be ignored.

• **Exclude**: Defines which pairs of bonded atoms should be excluded from non-bonded interactions.
  - Value 1: <STRING> – Allows you to choose between “1-2”, “1-3” and “1-4”.
    - 1-2: All interaction pairs of bonded atoms except the ones that separated with one bond, will be considered and modified using 1-4 parameters defined in parameter file.
    - 1-3: All interaction pairs of bonded atoms except the ones that separated with one or two bonds, will be considered and modified using 1-4 parameters de-
fined in parameter file.

- 1-4: All interaction pairs of bonded atoms except the ones that separated with one, two or three bonds, will be considered using nonbonded parameters defined in parameter file.

**Note:**
If no value was detected, by default “1-3” will be considered.

- **Potential**: Defines the potential function type to calculate nonbonded interaction energy and force between atoms.
  - Value 1: <STRING> – Allows you to pick between “VDW”, “SHIFT” and “SWITCH”
    - VDW: Nonbonded interaction energy and force calculated based on n-6 (Lennard-Johns) equation. This function will be discussed in details in Intermolecular energy and Virial calculation section.

      Example:
      
      `# SIMULATION CONDITION
      Temperature     200.00
      Potential       VDW
      LRC             true
      Rcut            10
      Exclude         1-4`

      - SHIFT: This option forces the potential energy to be zero at Rcut distance. This function will be discussed in details in Intermolecular energy and Virial calculation section.

      Example:
      
      `# SIMULATION CONDITION
      Temperature     200.00
      Potential       SHIFT
      LRC             false
      Rcut            10
      Exclude         1-4`

      - SWITCH: This option smoothly forces the potential energy to be zero at Rcut distance and start modifying the potential at Rswitch distance. This function will be discussed in details in Intermolecular energy and Virial calculation section.

    - **Rswitch**: Sets a distance which nonbonded interaction energy will be truncated smoothly from to cutoff distance.
• Value 1: <DOUBLE> – Define switch distance. If “SWITCH” function is chosen, Rswitch need to be defined otherwise program will be terminated.

Example:

```
#############################
# SIMULATION CONDITION
#############################
Temperature     200.00
Potential     SWITCH
Rswitch     7
Rcut     10
LRC     false
Exclude     1-3
```

Note/Warning:
In CHARMM forcefield the 1-4 interaction is need to be considered. Choosing “Exclude 1-3” will modify 1-4 interaction based on 1-4 parameter in parameter file. If a forcefield type was used which doesn’t require 1-4 interaction such as TraPPE, either “exclude 1-4” need to be chosen or 1-4 parameter need to be assigned to zero in parameter file.

• **Ewald**: Defines to consider PME for electrostatic calculation.
  
  o Value 1: <BOOLEAN> – True if PME calculation need to be considered, false if not.

  Warning:
PME calculation will be implemented in future and currently GOMC unable to support charged atoms. Ewald need to be set to false, otherwise program will be terminated.

• **RunSteps**: Sets total number of steps to run for (one move is performed for each step) (cycles = this value / number of molecules in the system)
  
  o Value 1: <ULONG> – total run steps

• **EqSteps**: Sets the number of steps necessary to equilibrate the system; averaging will begin at this step.
  
  o Value 1: <ULONG> – equilibration steps

• **AdjSteps**: Sets the number of steps per adjustment to the maximum constants associated with each move (e.g. maximum distance in xyz to displace, the maximum volume in Å³ to swap, etc.)
  
  o Value 1: <ULONG> – number of steps per move adjustment
- **ChemPot**: (FOR grand canonical (GC) ensemble runs only): Chemical potential to run the simulation at.
  - Value 1: <STRING> – The resname to apply this chemical potential w.r.t.
  - Value 2: <DOUBLE> – The chemical potential value in degrees Kelvin (should be negative).

  Note:
  For binary systems include multiple copies of the tag, one per residue kind.

  Example:
  
  # STEPS
  RunSteps 25000000
  EqSteps 5000000
  AdjSteps 1000

<table>
<thead>
<tr>
<th>Mol. Name</th>
<th>Chem. Pot.(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChemPot</td>
<td>AR</td>
</tr>
<tr>
<td></td>
<td>-968</td>
</tr>
</tbody>
</table>

- **DisFreq**: Fractional percentage which displace move will happen
  - Value 1: <DOUBLE> – % displace

- **RotFreq**: Fractional percentage which rigid rotation move will happen
  - Value 1: <DOUBLE> – % rotate

- **VolFreq**: Fractional percentage which volume displacement move will happen
  - Value 1: <DOUBLE> – (FOR Gibbs ensemble runs only) % of volume swaps

- **SwapFreq**: Fractional percentage which particle swap move will happen
  - Value 1: <DOUBLE> – (FOR Gibbs and GC ensemble runs only) % of molecule swaps

Example:

# MOVE FREQUENCY
DisFreq 0.69
<table>
<thead>
<tr>
<th>RotFreq</th>
<th>0.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>VolFreq</td>
<td>0.01</td>
</tr>
<tr>
<td>SwapFreq</td>
<td>0.20</td>
</tr>
</tbody>
</table>

**Note:**
All move percentages should add up to 1.0, otherwise program would be terminated.

- **BoxDim**: Defines the axis lengths of simulation box. This tag may occur multiple times (It occurs once for NVT, but twice for Gibbs ensemble or GC ensemble).
  - Value 1: `<INTEGER>` – sets box number (first box is box ‘0’)
  - Value 2: `<DOUBLE>` – x-axis length in Angstroms
  - Value 3: `<DOUBLE>` – y-axis length in Angstroms
  - Value 4: `<DOUBLE>` – z-axis length in Angstroms

**Note:**
If number of defined boxes were not compatible to simulation type, program will be terminated.

Example (NVT ensemble):
```
###################################
# BOX DIMENSION #, X, Y, Z
###################################
BoxDim 0   40.00 40.00 40.00
```

Example (Gibbs ensemble or GC ensemble):
```
###################################
# BOX DIMENSION #, X, Y, Z
###################################
BoxDim 0   106.60 106.60 106.60
BoxDim 1   176.60 176.60 176.60
```

- **CBMC_First**: (FOR Gibbs and GC ensemble runs only, currently) Number of CBMC trials to choose the first seed position (Lennard-Jones trials for first seed growth)
  - Value 1: `<INTEGER>` – Number of initial insertion sites to try

- **CBMC_Nth**: (FOR Gibbs and GC ensemble runs only, currently) Number of CBMC trials to
choose the later seed positions (Lennard-Jones trials for first seed growth)

- Value 1: <INTEGER> – Number of LJ trials for growing later atom positions

- **CBMC_Ang**: (FOR Gibbs and GC ensemble runs only, currently) Number of CBMC bending angle trials to perform for geometry (per the coupled-decoupled CBMC scheme)
  
  - Value 1: <INTEGER> – Number of trials per angle

- **CBMC_Dih**: (FOR Gibbs and GC ensemble runs only, currently) Number of CBMC dihedral angle trials to perform for geometry (per the coupled-decoupled CBMC scheme)
  
  - Value 1: <INTEGER> – Number of trials per dihedral.

Example:

```
# CBMC TRIALS

CBMC_First 10
CBMC_Nth 4
CBMC_Ang 100
CBMC_Dih 10
```

**Output Controls**

This section contains all the values that control output in the control file. For example certain variables control the naming of files dumped of the block averaged thermodynamic variables of interest, the PDB files, etc.

- **OutputName**: Unique name for simulation used to name the block average, fluctuation, PDB and PSF output files.
  
  - Value 1: <STRING> – unique phrase to identify this system

Example:

```
# OUTPUT FILE NAME

OutputName ISB_T_270.00_K
```

- **CoordinatesFreq**: Controls output of PDB file (coordinates) and PRNG state. If PDB dumping was enabled one file for NVT, two files for Gibbs ensemble or GC ensemble will be dumped.
  
  - Value 1: <BOOLEAN> – “true” enables dumping these files; “false” disables dumping.
- **Value 2:** `<ULONG>` – steps per dump PDB frame and it should be less or equal to `RunSteps`. If this keyword could not be found in configuration file, this value will be assigned by default in order to dump 10 frames.

  **Note/Warning:**
  The PDB file contains an entry for every `ATOM`, in all boxes read in. This allows VMD (which requires a constant number of atoms to properly parse frames (with a bit of help)). Atoms that are not currently in a specific box are given the coordinate 0.00, 0.00, 0.00. The occupancy value corresponds to the box a molecule is currently in (e.g. 0.00 for box 0; 1.00 for box 1).

  **Note:**
  At the beginning of simulation a merged PSF file contains an entry for every `ATOM`, in all boxes will be dumped. It also contains all the topology for every molecule in both boxes, corresponding to the merged PDB format. Loading PDB files into merged PSF file in VMD allow user to visualize and analyze the results.

- **RestartFreq:** Controls output of last state of simulation at specified step in PDB files (coordinates) that contain “_restart” phrase. If PDB dumping was enabled one file for NVT, two files for Gibbs ensemble or GC ensemble will be dumped.
  - Value 1: `<BOOLEAN>` – “true” enables dumping these files; “false” disables dumping.
  - Value 2: `<ULONG>` – steps per dump last state of simulation to PDB files and it should be less or equal to `RunSteps`. If this keyword could not be found in configuration file, by default `RunSteps` value will be assigned to it.

  **Note/Warning:**
  The restart PDB file contains only `ATOM` that exist in each boxes at specified steps. This allows user to use psfgen and tcl to build new PSF file and load it back to `RESTART` the simulation.

- **ConsoleFreq:** Controls the output to STDIO (“the console”) of messages such as acceptance statistics, averages, and run timing info.
  - Value 1: `<BOOLEAN>` – “true” enable message printing; “false” disables dumping.
  - Value 2: `<ULONG>` – number of steps per print. If this keyword could not be found in configuration file, this value be assigned by default in order to dump 1000 output for `RunSteps` greater than 1000 steps and 100 output for `RunSteps` less than 1000 steps.
- **BlockAverageFreq**: Controls the output of block averages. Block averages are averages of thermodynamic values of interest for chunks of the simulation (for post processing of averages or std. dev. In those values).
  
  - Value 1: <BOOLEAN> – “true” enable printing block average; “false” disables it.
  
  - Value 2: <ULONG> – number of steps per block average output file. If this keyword could not be found in configuration file, this valued be assigned by default in order to dump 100 output.

- **FluctuationFreq**: Controls the fluctuation logs. Fluctuations are a log of instantaneous values for those quantities.
  
  - Value 1: <BOOLEAN> – “true” enable printing fluctuation; “false” disables it.
  
  - Value 2: <ULONG> – number of steps per fluctuation output file. If this keyword could not be found in configuration file, this valued be assigned by default in order to dump 1000 output for `RunSteps` greater than 1000 steps and 100 output for `RunSteps` less than 1000 steps.

- **HistogramFreq**: Controls the histograms. Histograms are a binned listing of observation frequency for a specific thermodynamic variable. In this code they also control the output of a file containing energy/particle samples, it only will be used in GC ensemble simulations for histogram reweighting purposes.
  
  - Value 1: <BOOLEAN> – “true” enable printing histogram; “false” disables it.
  
  - Value 2: <ULONG> – number of steps per histogram output file. If this keyword could not be found in configuration file, this valued be assigned by default in order to dump 1000 output for `RunSteps` greater than 1000 steps and 100 output for `RunSteps` less than 1000 steps.

---

```
# STATISTICS Enable, Freq.

CoordinatesFreq  true  10000000
RestartFreq      true  10000000
ConsoleFreq      true  1000000
BlockAverageFreq false  100000
FluctuationFreq  false  10000
HistogramFreq    true  10000
```

Next section controls the output of the energy/particle sample file and the distribution file for particle counts, commonly referred to as the “histogram” output. This section
only requires if GC ensemble simulation was used.

- **DistName**: Sets short phrase to naming particle distribution file.
  - Value 1: <STRING> – Short phrase which will be combined with RunNumber and RunLetter to use in the name of the binned histogram for particle distribution.

- **HistName**: Sets short phrase to naming energy sample file.
  - Value 1: <STRING> – Short phrase, which will be combined with RunNumber and RunLetter to use in the name of the energy/particle count sample file.

- **RunNumber**: Sets a number, which is a part of DistName and HistName file name.
  - Value 1: <UINT> – Run number to use in the above file names.

- **RunLetter**: Sets a letter, which is a part of DistName and HistName file name.
  - Value 1: <CHAR> – Run letter to use in above file names.

- **SampleFreq**: Controls histogram sampling frequency.
  - Value 1: <UINT> – the number of steps per histogram sample.

Example:
```
# OutHistSettings
DistName dis
HistName his
RunNumber 5
RunLetter a
SampleFreq 200
```

- **OutEnergy***, **OutPressure***, **OutMolNumber***, **OutDensity***, **OutVolume***: Enables/disables for specific kinds of file output for tracked thermodynamic quantities
  - (*) = NVT ensemble  (**) = Gibbs ensemble  (***) = GC ensemble
  - Value 1: <BOOLEAN> – “true” enable message output of block averages of this tracked parameter (and in some cases such as entry, components); “false” disables it.
  - Value 2: <BOOLEAN> – “true” enable message output of a fluctuation log for this tracked parameter (and in some cases such as entry, components); “false” disables it.
Value 3: <BOOLEAN> – “true” enable message output of a histogram of observation frequency for values of this tracked parameter (and in some cases such as entry, components); “false” disables it.

Example:

```plaintext
# ENABLE: BLK AVF., FLUCK., HIST.
OutEnergy true true true
OutPressure true true true
OutMolNum true true true
OutDensity true true true
```

**Note/Warning:**
If any of BlockAverageFreq, FluctuationFreq or HistogramFreq was set to false and in this section was set to true, it will be ignored and the properties will not be printout.

**Sample Files**

Here’s a sample of how a typical NVT ensemble control file might be expected to look:

```plaintext
# GOMC beta 1.00 - NVT Control example

# ENABLE, STEP
Restart false

# KIND {RESTART, RANDOM, INTSEED}
PRNG RANDOM

# FORCE FIELD
ParaTypeCHARMM true
ParaTypeEXOTIC false
Parameters Par.MODEL_NAME.inp

# INPUT PDB FILES
Coordinates 0 STEP3_START_XX_PHA_BOX_0.pdb
```
# INPUT PSF FILES
Structure 0 STEP3_START_xx_PHA_BOX_0.psf

# SIMULATION CONDITION
Temperature 200.00
Potential VDW
LRC true
Rcut 10
Exclude 1-4

# ELECTROSTATIC
Ewald false

# STEPS
RunSteps 25000000
EqSteps 5000000
AdjSteps 1000

# MOVE FREQUENCY
DisFreq 0.90
RotFreq 0.10

# BOX DIMENSION #, X, Y, Z
BoxDim 0 35.00 35.00 35.00

# OUTPUT FILE NAME
OutputName AR_200_00_k_2220_r5

# STATISTICS ENABLE, FREQ.
CoordinatesFreq true 1000000
RestartFreq true 1000000
ConsoleFreq true 1000000
Here’s a sample of how a typical Gibbs ensemble control file might be expected to look:

```plaintext
#================================
# GOMC beta 1.00 – Gibbs ensemble Control example =
#================================

# ENABLE, STEP
Restart false

# KIND {RESTART, RANDOM, INTSEED}
PRNG INTSEED Random_Seed 49

# FORCE FIELD
ParaTypeCHARMM false
ParaTypeEXOTIC true
Parameters Par_MODEL_NAME.inp

# INPUT PDB FILES
Coordinates 0 STEP3_START_XX_PHA_BOX_0.pdb
Coordinates 1 STEP3_START_XX_PHA_BOX_1.pdb

# INPUT PSF FILES
Structure 0 STEP3_START_XX_PHA_BOX_0.psf
Structure 1 STEP3_START_XX_PHA_BOX_1.psf

# GEMC TYPE (DEFAULT IS NVT_GEMC)
```
GEMC NPT
Pressure 2.50

# SIMULATION CONDITION
Temperature 200.00
Potential SWITCH
LRC false
Rswitch 7
Rcut 10
Exclude 1-4

# ELECTROSTATIC
Ewald false

# STEPS
RunSteps 25000000
EqSteps 5000000
AdjSteps 1000

# MOVE FREQUENCY
DisFreq 0.69
RotFreq 0.10
VolFreq 0.01
SwapFreq 0.20

# BOX DIMENSION #, X, Y, Z
BoxDim 0 35.00 35.00 35.00
BoxDim 1 45.00 45.00 45.00

# CBMC TRIALS
CBMC_First 10
CBMC_Nth 4
CBMC_Ang 100
CBMC_Dih 10

# OUTPUT FILE NAME
Here’s a sample of how a typical grand canonical ensemble control file might be expected to look:

```
# GOMC beta 1.00 – Grand canonical ensemble Control example =
#===============================================

# ENABLE, STEP
Restart false

# KIND {RESTART, RANDOM, INTSEED}
PRNG INTSEED
Random_Seed 49

# FORCE FIELD
ParaTypeCHARMM false
ParaTypeEXOTIC true
Parameters Par_MODEL_NAME.inp

# INPUT PDB FILES
Coordinates 0 STEP3_START_XX_PHA_BOX_0.pdb
Coordinates 1 STEP3_START_XX_PHA_Reserv_BOX_1.pdb
```

```
Structure 0  STEP3_START_XX_PHA_BOX_0.psf
Structure 1  STEP3_START_XX_PHA_Reserv_BOX_1.psf

# SIMULATION CONDITION
Temperature 200.00
Potential SHIFT
LRC false
Rcut 10
Exclude 1-4

# ELECTROSTATIC
Ewald false

# STEPS
RunSteps 25000000
EqSteps 5000000
AdjSteps 1000

# MOVE FREQUENCY
DisFreq 0.69
RotFreq 0.10
VolFreq 0.01
SwapFreq 0.20

# BOX DIMENSION #, X, Y, Z
BoxDim 0 35.00 35.00 35.00
BoxDim 1 45.00 45.00 45.00

# CBMC TRIALS
CBMC_First 10
CBMC_Nth 4
CBMC_Ang 100
CBMC_Dih 10
# Mol. Name     Chem. Pot.
ChemPot   AR     -2230

# OUTPUT FILE NAME
OutputName  AR_200_00_K_2220

# STATISTICS ENABLE, FREQ.
CoordinatesFreq  true  1000000
RestartFreq     true  1000000
ConsoleFreq     true  1000000
BlockAverageFreq true  1000000
FluctuationFreq true  1000000
HistogramFreq   true  1000000

# OutHistSettings
DistName  dis
HistName  his
RunNumber 5
RunLetter a
SampleFreq 200

# ENABLE: BLK AVF., FLUCK., HIST.
OutEnergy  true  true  true  true
OutPressure true  true  true
OutMolNum   true  true  true
OutDensity  true  true  true

GOMC’s Output Files, Terminal Output
GOMC currently supports several kinds of output:

- STDIO (“console”) Output
- File Output
  - PDB
  - PSF
  - Block Averages
Console Output
A variety of useful information relating to variable averages, acceptance rates, file I/O messages warnings, and other kinds of information is printed formatted to the STDIO, which in Linux will typically be displayed in the terminal. This output can be redirected into a log file in Linux using the ‘>’ operator.

The first section of this console output typically includes some intro info relating to the forcefield read. This output is important as it may contain text relating to issues encountered if there was an error in the current run (e.g. a bad parameter, etc.)
Next some info on the system's starting configuration will print:
For most of the run info on the variables and move statistics will be printed, e.g.:

```
STEP 9999 of 40000000 (0.0249975% done)

--------

Volume: 361095 A³
Tot # mol: 946
Molecule Type ISB
Number: 946
Density: 0.232856 g/ml

Energy (in K): total: -491491
  inter: -713432; inter (tail corr.): -74880.3
  intra (bonded): 296821; intra (nonbonded): 0

Displace (Box 0) -- tries: 3329; # Accept: 1386; % Accept: 41.0333; Max Amt.: 0.919088
  Rotate (Box 0) -- tries: 420; # Accept: 241; % Accept: 57.381; Max Amt.: 3.12267
  Molecule Transfer (Box 0 -> Box 1) -- tries: 1265; # Accept: 562; % Accept: 44.4269
  Volume Transfer (Box 0 -> Box 1) -- tries: 82; # Accept: 54; % Accept: 65.8537; Max Amt.: 6834.3

--------

Volume: 269650 A³
Tot # mol: 1054
Molecule Type ISB
Number: 1054
Density: 0.275204 g/ml

Energy (in K): total: -619525
  inter: -855439; inter (tail corr.): -90802.5
  intra (bonded): 326716; intra (nonbonded): 0

Displace (Box 1) -- tries: 3226; # Accept: 1293; % Accept: 40.0806; Max Amt.: 0.879687
  Rotate (Box 1) -- tries: 463; # Accept: 275; % Accept: 59.3952; Max Amt.: 3.14159
  Molecule Transfer (Box 1 -> Box 0) -- tries: 1215; # Accept: 508; % Accept: 41.8107
  Volume Transfer (Box 1 -> Box 0) -- tries: 0; # Accept: 0; % Accept: 0; Max Amt.: 300

--------

Energy (in K): total: -1.11102e+06
  inter: -1.56887e+06; inter (tail corr.): -165683
  intra (bonded): 623538; intra (nonbonded): 0
Steps/sec.: 178.955
```

Note:
It’s important to watch the acceptance rates and adjust the move percentages and CBMC trial amounts to get the desired rate of move acceptance.
At the end of the run timing information and other wrapup info will print, such as any failures encountered in the energetic consistency checks:

![Simulation Output](image)

**PDB and PSF Files**

The PDB and PSF output (merging of atom entries) has already been mentioned/explained in previous sections. To recap: the PDB file’s `ATOM` entries’ occupancy is used to represent the box the molecule is in in the current frame. All molecules are listed in order they were read in. i.e. if box 0 has \(1..N_1\) molecules and box 1 has \(1..N_2\) molecules, then all of the molecules in box 0 are listed first, then all the molecules in box 1, i.e. \(1..N_1, N_1+1..N_1+N_2\). PDB frames are written as standard PDBs to consecutive file frames.

To visualize you’ll need to open the PDB and PSF outputted files in VMD and use the command:

```
pbc join connected
```

In the TKConsole in order to get the current frame to obey the periodicity. Some peculiarities are still be observed, so please review the PDB file if anything looks odd (e.g. bonds will sometimes not show up in later frames). The GOMC team is in the process of improving VMD compatibility; but this may require changes to VMD to allow the files outputted to be properly read.

For Gibbs ensemble, the same PSF will be used to visualize either box, i.e.

```
vmd ISB_T_330.00_K_Run_0_MIE_BOX_0.pdb ISB_T_330.00_K_Run_0_MIE_merged.pdb
```

...for visualizing the first box and...

```
vmd ISB_T_330.00_K_Run_0_MIE_BOX_1.pdb ISB_T_330.00_K_Run_0_MIE_merged.pdb
```

... for visualizing the second box.

**Block Output Files**

GOMC tracks a number of thermodynamics variables of interest during the simulation, some independent, others dependent on each other. The variables are tracked by this module:

- Density (g/ml)
- Energy
  - Intermolecular
  - intramolecular bonded
  - Intramolecular nonbonded
It is convenient to average the block file using simple shell utilities in Linux:

```
alias avg 'awk -v start_ln=\!:2 -v column=\!:3 """"BEGIN {n=0; sum=0; std_dev=\n0} NR>start_ln { sum+=$column; result[n]=$column; n++} END {
    avg=sum/n; sum=0; \n    for(x=1;x<=n;x++){ sum+=(result[x]-avg)^2}; std_dev=sqrt(sum/n);
    print avg" "std_dev }"""" \!:1'
```

```
alias avgd 'awk -v start_ln=\!:2 -v column=\!:3 """"BEGIN {n=0; sum=0; std_dev=\n0} NR>$start_ln { result[n]=$column*1000; sum+=result[n]; n++} END {
    avg=sum/n} ; sum=0; for(x=1;x<=n;x++){ sum+=(result[x]-avg)^2};
    std_dev=sqrt(sum/n); print avg" "std_dev }"""" \!:1'
```

To call this, make sure its directory is in your path and type “./

On the command file the name of the file goes first.

```
avg Blk_Pressure_T_330.00_K_Run_0_MIE.dat 100 2
```

This should display the average, skipping the first 100 lines of the file. Adjust the # of skipped lines as appropriate based on when the system is found to be equilibrated, and adjust the fourth parameter for the desired box (with 1 being box 0, 2 being box 1).

**Putting it all Together: Running a GOMC Simulation**

Strongly recommended to download the test system provided at [http://gomc.eng.wayne.edu/downloads.aspx](http://gomc.eng.wayne.edu/downloads.aspx) and run different simulation type in order to get familiar with different parameter and configuration files (in.dat).
To recap the previous examples, a simulation of isobutane will be completed for a single temperature point on the saturated vapor liquid coexistence curve. Please refer to the attached reference directories if you get stuck.

Files for this simulation will be put in the empty folder “1-Isobutane-UserGenerated”. If errors are encountered during the building of the system please refer to the folders “2-Isobutane-Reference/1_ScriptsComplete/” and “1-Isobutane-Reference/2_FullyBuilt/” for the proper files and results at various stages of the build.

The general plan for running the simulation is:

1. Build GOMC (if not done already)
2. Copy GOMC GEMC executable to build directory
3. Create scripts, PDB, and topology file to build the system, plus in.dat file and parameter files to prepare for runtime
4. Build finished PDBs and PSFs using the simulation.
5. Run the simulation in the terminal.
6. Analyze the output.

Please complete steps 1 and 2 and then traverse to the directory, which should now have a single file “GOMC_Serial_GEMC” in it.

Next, six files need to be made:

- PDB file for isobutane
- Topology file describing isobutane residue
- Two *.inp packmol scripts to pack two system boxes
- Two TCL scripts to input into PSFGen to generate the final configuration

isobutane.pdb:

```
REMARK 1 File created by GaussView 5.0.8
ATOM  1  C1   ISB   1   0.911 -0.313  0.000  C
ATOM  2  C2   ISB   1   1.424 -1.765  0.000  C
ATOM  3  C3   ISB   1  -0.629 -0.313  0.000  C
ATOM  4  C4   ISB   1   1.424  0.413 -1.257  C
END
```

Top_Branched_Alkane.inp:

```
* Custom top file -- branched alkanes
*
! MASS  1  CH3  15.035  C  !
MASS  3  CH1  13.019  C  !
```
RESI ISB 0.00 ! isobutane - TraPPE
GROUP
ATOM C1 CH1 0.00 ! C3
ATOM C2 CH3 0.00 ! C2-C1
ATOM C3 CH3 0.00 ! C4
ATOM C4 CH3 0.00 !
BOND C1 C2 C1 C3 C1 C4
PATCHING FIRS NONE LAST NONE
END

pack_box_0.inp:

tolerance 3.0
filetype pdb
output STEP2_ISB_packed_BOX_0.pdb
structure isobutane.pdb
number 1000
inside box 0. 0. 0. 68.00 68.00 68.00
end structure

pack_box_1.inp:

tolerance 3.0
filetype pdb
output STEP2_ISB_packed_BOX_1.pdb
structure isobutane.pdb
number 1000
inside box 0. 0. 0. 68.00 68.00 68.00
end structure

build_box_0.inp:

psfgen << ENDMOL
topology ./Top_Branched_Alkane.inp
segment ISB {
  pdb ./STEP2_ISB_packed_BOX_0.pdb
  first none
  last none
}
coordpdb ./STEP2_ISB_packed_BOX_0.pdb ISB
writepsf ./STEP3_START_ISB_sys_BOX_0.psf
writepdb ./STEP3_START_ISB_sys_BOX_0.pdb

build_box_1.inp:
These files can be created with a standard Linux or Windows text editor. Please also copy a packmol executable into the working directory. The resulting folder can be compared against the "Expected_Results/Build_Scripts_and_Model" directory in the sample code folder.

Once those files are created, run:

```
./packmol < pack_box_0.inp
./packmol < pack_box_1.inp
```

This will create the intermediate PDBs. Please examine them for correctness (the results obtained can be compared against the “Expected_Results/PDB_Intermediates” directory in the sample code folder.

Then run the PSFGen scripts to finish the system

```
./build_box_0.inp
./build_box_1.inp
```

Note, you probably will have to first make your scripts executable using:

```
chmod a+x build_box_*.inp
```

This will create the intermediate PDBs. Please examine them for correctness (the results obtained can be compared against the “Expected_Results/Complete_PDB_and_PSF” directory in the sample code folder.

To run the code a few additional things will be needed:

- A GOMC Gibbs ensemble executable
- A control file
- Parameter files.

Enter the control file (**in.dat**) in the text editor in order to modify it. Example files for different simulation type can be find in previous section.
Once these four files have been added to the output directory the simulation should be ready to run. The files and directory can be compared to “Expected_Results/Ready_To_Run” directory in the sample code folder.

Assuming the code is named GOMC_Serial_GEMC run using:

```
./GOMC_Serial_GEMC > out_ISB_T_330.00_K_RUN_0.log &
```

Progress can be monitored with the tail command:

```
tail -f out_ISB*log
```

When the run is done, please compare the results to those found in “Expected_Results/Simulation_Output” directory in the sample code folder. Use the “avg” alias, added in the previous section to determine the equilibrated block averages.

Congratulations, a single phase coexistence point on the saturated vapor liquid curve has been examined using GOMC operating in the Gibbs ensemble.

This process can be repeated additional times to produce new results (saving the old results aside) to produce averages. It is recommended that at least three simulations be run per point, although for larger/longer simulations sometimes it is appropriate to run only a single simulation.

![Graph showing Isobutane TraPPE FF](image_url)
Repeating this process for multiple temperatures will allow you to obtain the following results. This data is contained in the folder “Expected_Results/TraPPE_Isobutane_VLE_Graphs”.

**Intermolecular Energy and Virial function**

- **VDW**: This option calculates potential energy without any truncation.
  
  - Potential calculation: Interactions between atoms can be modeled with an $n-6$ potential, a Mie potential in which the attractive exponent is fixed. The Mie potential can be viewed as a generalized version of the 12-6 Lennard-Jones potential,

    $$E_{ij} = C_{n_{ij}}\varepsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^6 \right]$$

    where $r_{ij}$, $\varepsilon_{ij}$, and $\sigma_{ij}$ are, respectively, the separation, well depth, and collision diameter for the pair of interaction sites $i$ and $j$. The constant $C_n$ is a normalization factor used such that the minimum of the potential remains at $-\varepsilon_{ij}$ for all $n_{ij}$. In the 12-6 potential, $C_n$ reduces to the familiar value of 4.

    $$C_{n_{ij}} = \left( \frac{n_{ij}}{n_{ij} - 6} \right) \left( \frac{r_{ij}}{6} \right)^{6/(n_{ij} - 6)}$$

  - Virial calculation: Virial is basically the negative derivative of energy with respect to distance multiply by distance.

    $$W_{ij} = -\frac{dE_{ij}}{dr} \times \frac{r_{ij}}{r_{ij}}$$

    Using n-6 LJ potential defined above:

    $$W_{ij} = 6 \times C_{n_{ij}}\varepsilon_{ij} \left[ \frac{n_{ij}}{2} \times \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] \times \frac{r_{ij}}{r_{ij}^2}$$

**Note:**

This option only evaluates the energy up to specified $R_{cut}$ distance, tail correction to energy and pressure can be specified to account for infinite cutoff distance.

- **SHIFT**: This option forces the potential energy to be zero at $R_{cut}$ distance.
Potential calculation: Interactions between atoms can be modeled with an \( n - 6 \) potential,

\[
E_{ij}^{(\text{shift})} = C_{n_{ij}} \varepsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] - C_{n_{ij}} \varepsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r_c} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_c} \right)^6 \right]
\]

where \( r_{ij}, \varepsilon_{ij}, \) and \( \sigma_{ij} \) are, respectively, the separation, well depth, and collision diameter for the pair of interaction sites \( i \) and \( j \). The constant \( C_n \) is a normalization factor used such that the minimum of the potential remains at \( -\varepsilon_{ij} \) for all \( n_{ij} \). In the 12-6 potential, \( C_n \) reduces to the familiar value of 4.

\[
C_{n_{ij}} = \left( \frac{n_{ij}}{n_{ij} - 6} \right) \left( \frac{r_{ij}}{6} \right)^{(n_{ij}-6)}
\]

Virial calculation: Virial is basically negative derivative of energy with respect to distance multiply by distance.

\[
W_{ij} = -\frac{dE_{ij}}{dr} \times \frac{\vec{r}_{ij}}{r_{ij}}
\]

Using SHIFT potential function defined above:

\[
W_{ij}^{(\text{shift})} = 6 \times C_{n_{ij}} \varepsilon_{ij} \left[ \frac{n_{ij}}{2} \times \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] \times \frac{\vec{r}_{ij}}{r_{ij}^2}
\]

**Figure 1:** Graph of van der Waals potential with and without the application of the SHIFT function. With the SHIFT function active, the potential by forced reduced to 0.0 at the Rcut distance. With the SHIFT function, there is a discontinuity where the potential is truncated.
• **SWITCH:** This option smoothly forces the potential energy to be zero at Rcut distance and start modifying the potential at Rswitch distance.

  - Potential calculation: Interactions between atoms can be modeled with an n − 6 potential,
    \[ E_{ij}(\text{switch}) = C_{n_{ij}}\varepsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] \times F_E \]
    where \( r_{ij}, \varepsilon_{ij}, \) and \( \sigma_{ij} \) are, respectively, the separation, well depth, and collision diameter for the pair of interaction sites \( i \) and \( j \). The constant \( C_n \) is a normalization factor used such that the minimum of the potential remains at \(-\varepsilon_{ij}\) for all \( n_{ij} \). In the 12-6 potential, \( C_n \) reduces to the familiar value of 4.
    \[ C_{n_{ij}} = \left( \frac{n_{ij}}{n_{ij} - 6} \right) \left( \frac{n_{ij}}{6} \right)^{6/(n_{ij}-6)} \]
    The factor \( F_E \) is defined as:
    \[
    F_E = \begin{cases}
      1 & r_{ij} \leq r_{\text{switch}} \\
      \frac{1}{(r_{\text{cut}}^2 - r_{ij}^2)^3} & r_{\text{switch}} < r_{ij} < r_{\text{cut}} \\
      0 & r_{ij} \geq r_{\text{cut}}
    \end{cases}
    \]
  - Virial calculation: Virial is basically negative derivative of energy with respect to distance multiply by distance.
    \[ W_{ij} = -\frac{dE_{ij}}{dr} \times \frac{\vec{r}_{ij}}{r_{ij}} \]
    Using SWITCH potential function defined above:
    \[
    W_{ij}(\text{switch}) = 6 \times C_{n_{ij}}\varepsilon_{ij} \left[ \frac{n_{ij}}{2} \times \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] \times F_E \times \frac{\vec{r}_{ij}}{r_{ij}^2} - C_{n_{ij}}\varepsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{n_{ij}} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] \times F_W
    \]
    The factor \( F_W \) is defined as:
    \[
    F_W = \begin{cases}
      1 & r_{ij} \leq r_{\text{switch}} \\
      \frac{12(r_{\text{cut}}^2 - r_{ij}^2) \times (r_{\text{switch}}^2 - r_{ij}^2)}{(r_{\text{cut}}^2 - r_{\text{switch}}^2)^3} & r_{\text{switch}} < r_{ij} < r_{\text{cut}} \\
      0 & r_{ij} \geq r_{\text{cut}}
    \end{cases}
    \]
Figure 2: Graph of van der Waals potential with and without the application of the SWITCH function. With the SWITCH function active, the potential is smoothly reduced to 0 at the Rcut distance.

How to Get Help/Technical Support
Please send an email to gomc@eng.wayne.edu