CoSIMS User's Manual

Christopher A. Myers

February 6, 2019

Contents

1 Installation

1.1 Obtaining the code

To obtain the program from the GitHub repository, simply type

```
$ git clone https://github.com/ChristopherAMyers/CoSIMS.git
```

in the directory that you want to save the code program in. Alternatively, you can download the program as a `.zip` file and extract the contents manually.

1.2 Building

To build the program, `g++` version 4.8.1 or above is required. Other compilers may work, but have not been tested yet. Additionally, CoSIMS is designed around the OpenMP multi-threading library for computing the trajectories for the CCS integrals in parallel. Thus, OpenMP must be available to your compiler. CoSIMS has also only been designed and tested on linux systems, so your milage may vary if use don a Windows environment.

To build the program, simply type `make` in the CoSIMS directory that contains the file `Makefile`

```
$ make
```

or

```
$ make -j
```

to enable multiple files to be compiled at once. This process should only take a few minutes, as there are not many code files to build and link. CoSIMS runs as a standalone binary and does not integrate into the operation system like other packages do. The program also compiles all libraries into a single, statically linked binary. The compiled binary will be saved in `/path/to/cosims/install/bin/cosims`

Compilation can also be done manually by executing the following command in the extracted directory:

```
$ g++ src/*.cpp -o cosims -std=c++11 -fopenmp -O3
```

2 Running CoSIMS

To run the program from the command line, only the molecule file is required as an input. To run, simply type

```
$ ./cosims -m moleculeFile.xxx
```

where `xxx` cna either be `pdb` or `mfj`. Please see section YYY for details. CoSIMS can also be called with additional command line arguments with the usage syntax being

```
$ ./cosims -m moleculeFile.xxx -[option] [value]
```

The next few sections will specify the possible molecule file types and the optional values for `[option]`.

2.1 Molecule file types

The `-m` option specifies the molecule file that is used to calculate the CCS. Currently, CoSIMS can only run one molecule at a time, and the ability to calculate multiple CCS for multiple structures will be added in later versions. CoSIMS can read `pdb` or `mfj` file types.

2.1.1 PDB File Types

For `pdb` file types, both `ATOM` and `HETATM` entries are used. Since CoSIMS can only take one structure at a time, the program will assume that all entries of this type are to be used in a single structure. This can cause problems when providing the program with `.pdb` files form the Protein Data Bank that contain multiple molecular models. Therefore, the files will have to be trimmed before feeding into CoSIMS.

Most `.pdb` files will utilize columns 78 and 79 to specify the element types. CoSIMS attempts to read this information, however, if these fields are blank, the program will attempt to use the atom name in columns 13-16. This will only take the first character of the atom into consideration and may not interpret the correct atom type. For example, a line in the `.pdb` that looks like this

```
ATOM      1 OH5'  C5      1       3.429 -7.861   3.641  1.00  0.00           H
```

would read the atom type to be a Hydrogen. However, if it were missing the last two columns,

```
ATOM      1 OH5'  C5      1       3.429 -7.861   3.641  1.00  0.00
```

Then CoSIMS will interpret the atom it as an Oxygen.

2.1.2 MFJ File Types

CoSIMS can also read `.mfj` file types, first proposed to be used with the MOBCAL CCS program produced by the Jarrold group [1, 2]. All standard syntax used in the original code still holds, with an additional feature. Charge types can now take integer (positive and negative) values. Similar to the string `equal` used in Mobcal, this will set a uniform charge distribution equal to that integer value provided.

2.2 Random Seed

The `-s` options can be used to specify the random number seed used by CoSIMS, which uses a 64-bit Mersenne Twister random number generator. If a seed is not provided bu the user, then the program will attempt to generate one based on the `random_device()` subroutine in the `c++ <random>` header library. Choosing the same seed for each calculation is not recommended as the Monte-Carlo evaluation of the CCS integrals will now use the same initial positions and velocities for all calculations. It should be noted that due to CoSIMS' implementation of a thread-save generator, using the same seed is not guaranteed to give the same results when using more than 1 OpenMP thread. This is because the work that is done on each thread is dynamically allocated, and each thread uses it's own sequence of random numbers starting at different positions in the generator's overall sequence.

2.3 Number of Threads

The `-n` option specifies the number of OpenMP threads to use for the integral evaluation. For best results, the number of threads must be less than or equal to the physical CPU cores available. As mentioned above in Section 2.2, when using multiple threads, identical program calls will not give the same CCS due to CoSIMS' attempt to make the random number generator thread safe. Each CCS should still agree within the standard deviation of the calculations.

2.4 Atom Types

The `-a` option allows the user to specify different atomic parameters, such as van der Waals radii σ, Lennard-Jones potential well depths ϵ, and atom masses. If one wanted to, for example, implement a coarse-grained model of their molecule, then an additional atom-type file can be provided with the `-a` option. Alternatively, this feature can be used as a way of implementing new force-fields.

Atom type files should follow the following rules:

1. Each line has the following properties: atom-name, atom-symbol, mass, vdW-Radii(σ), vdW-energy(ϵ), integer-mass
2. Each value is space delimited, and each line corresponds to a new atom type.
3. Comments lines start with the `#` character

As an example, an atom file could look like this:

```
#name       symbol  mass     sigma  epsilon     intMass
Carbon      C       12.010   2.043  0.03090114  12
Hydrogen    H       1.00794  2.043  0.01498936  1
Oxygen      O       15.9994  2.043  0.03090114  16
Phosphorus  P       30.9738  2.043  0.03113175  31
Nitrogen    N       14.0060  2.343  0.03090114  14
PhosNew     PN      35.9700  2.430  0.0300000   310
```

The Atom symbol (second column) can be used by `.pdb` file, as long as the symbol is specified in columns 78 and 79 of the file. Integer-mass numbers (sixth column) can be used as an identifier symbol for `.mfj` file types.

2.5 Input File

A powerful feature of CoSIMS is the ability for the user to change internal parameters of the program, without having to recompile the code from scratch. Examples include, but are not limited to, the trajectory time-step, number of trajectories, number of CCS integrals, and the temperature of the system. This is accomplished by providing an *optional* input file with the `-i` command line option. See Section 3 for details and syntax of this file.

3 Input File

The input file uses the following syntax to provide CoSIMS with additional operating settings:

```
program_option value
#comments use the '#' character
```

For example, the following text can be used as a proper input file.

```
#CoSIMS input file
name                 myName  #project name
dt                     0.01  #Verlet integration time step
temp                    298  #temprature in Kelvin
traj                     50  #trajectories per CCS integral
                             #(in thousands)
iter                     15  #number of CCS intergrals
threads                   1  #number of OpenMP threads to use
seed                  12345  #Mersenne Twister 64 bit seed
dispersion_cutoff      true  #LJ cut-off scheme used
dispersion_radius        40  #LJ cut-off radius to used in Ang
multipole              true  #use of multipole approximation
multipole_order           1  #uses dipole approximation
temp                    300  #overrides the previous option
```

Each possible program option that can be used in an input file will be listed in the following few sections of this manual along with a brief explanation of it's functionality. It should be noted that not every option is required to be included in an input file, and the user can choose which (if any) are appropriate for their calculations. Thus, all options, including the input file itself, is optional. As a result, each variable that can be changed will also have a default value. It should also be noted that if two input variables are listed, the one furthest down the file will be used. For example, a CCS calculation performed using the above input file would have a system temperature of 300 degree Kelvin, not 298.

3.1 Available Program Options

Option: `name`

Description:	Provides a project name for current calculation. This name will also be used to create the output file and/or the printed molecule file.
Data type:	string
Default value:	`log_charge`

Option: `dt`

Description:	Verlet Velocity integration time step in pico-seconds.
Data type:	float
Default value:	`0.01`

Option: `temp`

Description:	Temperature of the molecular system to be simulated in Kelvin
Data type:	float
Default value:	`298`

Option: `traj`

Description:	Changes the number of trajectories(scattering angles) to be calculated in each CCS integral in units of $\times 10^3$ trajectories.
Data type:	integer
Default value:	`50`

Option: `iter`

Description:	Number of iterations (CCS integrals) to average over.
Data type:	integer
Default value:	`10`

Option: `print_rate`

Description:	Program will print current CCS integral value and error every `print_rate` number of trajectories.
Data type:	integer
Default value:	`20000`
Note:	If `print_rate` is set to 0 or any number greater than the variable `traj`, then the current progress will not be printed.

Option: `print_mol`

Description:	Program will print centered and rotated molecule to an `XYZ` file.
Data type:	boolean
Default value:	`false`

Option: `dir`

Description:	Sets the file system directory to save all output files to.
Data type:	string
Default value:	`current-working-directory`

Option: `charge`

Description:	Molecule will be given a uniform charge distribution with a total charge equal to the value of `charge` in electric charge (e).
Data type:	integer
Default value:	`0`

Option: `threads`

Description:	Number OpenMP threads to use for CCS integral calculations. This will also override the `-n` command line option.
Data type:	integer
Default value:	`1`

Option: `seed`

Description:	Starting seed for Mersenne Twister 19937 64-bit random number generator. This will also override the `-s` command line option.
Data type:	long long integer
Default value:	`random_device()`
Note:	The default value is determined by the C++11 `random_device` library function. Please see the C++11 Standards Committee for further details.

Option: `dispersion_cutoff`

Description:	Enables the truncation of Lennard-Jones potential to extend no further than all atoms within a sphere of radius `dispersion_radius` centered at the gas atom's current trajectory position.
Data type:	boolean
Default value:	`true`

Option: `dispersion_radius`

Description:	Cut-off radius in Angstroms to be used if `dipersion_cutoff` is set to true.
Data type:	float
Default value:	`calculated via largest van der Waals radii in forcefield`

Option: `multipole`

Description:	Enables the multipole approximation for charged atoms that are distant from the gas atom's trajectory position.
Data type:	boolean
Default value:	`false`

Option: `multipole_radius`

Description:	If `multipole` is set to true, the electrostatic potential for all charged atoms outside a sphere of radius `multipole_radius` centered at the gas atom's position will be calculated via a multipole approximation.
Data type:	boolean
Default value:	`20.0`

Option: `multipole_order`

Description:	If `multipole` is set to true, this option sets the number of terms used in the multiple approximation. Each higher order term is slightly more expensive to compute than the last.
Example:	0 for monopole term. 1 for monopole and dipole terms. 2 for monopole, dipole, and quadrupole terms.
Data type:	integer
Default value:	`1`
Note:	If the charge is uniform and one uses dipole terms, only a monopole expansion will be used. This is because the geometric center of each atom cluster can be replaced by the center of charge, sending the dipole term to zero. Quadrupole terms do not seem to make much of a difference for uniform charge.

Option: `max_cluster_size`

Description:	Maximum radial distance from the center of each atom cluster to the farthest atom's position in each of their respective clusters.
Data type:	float
Default value:	`4.0`

References

[1] M. F. Mesleh, J. M. Hunter, A. A. Shvartsburg, G. C. Schatz, and M. F. Jarrold. Structural information from ion mobility measurements: effects of the long-range potential. *The Journal of Physical Chemistry*, 100(40):16082–16086, 1996.

[2] Alexandre A. Shvartsburg and Martin F. Jarrold. An exact hard-spheres scattering model for the mobilities of polyatomic ions. *Chemical Physics Letters*, 261(1):86 – 91, 1996.