User Manual for Monte Carlo Code

Tristan Truttmann

December 26, 2016

1 Improvements in 0.2b

Version 0.2b features a major bug fix and one additional feature. Version 0.1b contains a major bug that will produce inappropriate results in version 0.1b (It will produce energy fluctuations much larger than kT); this is fixed in this version. Version 0.2b also now has the ability to automatically attach the admolecule to the adsorbent when the two are supplied in different `.xyz` files. The user may now either supply the initial geometry of the admolecule already attached to the adsorbent in the file `SiestaFiles/InputGeom.xyz` as long as she supplies the `AdmoleculeSize` variable in the `MC_Pref.py` file, or she may alternatively supply the adsorbent geometry in the `adsorbent.xyz` file and the admolecule geometry in the `admolecule.xyz` file. If the two are supplied in seperate files, the `AdmoleculeSize` variable no longer need be specified. However, in order to use the automatic admolecule attachment feature, *the user must have* the python package `statsmodels`. If the user supplies all three of the `.xyz` files mentioned, an error will be raised. This automatic attachment offers the advantage of being able to screen a large number of admolecule-adsorbent combinations without having to take the time to manually dock the admolecules. If the user needs to specify a particular side to dock the admolecule onto, that can be done with the `RightSide` numpy array in the `MC_Pref.py` file. There is also one change to the `SiestaFiles/template.fdf` requirements. The `NumberOfAtoms` field must still be included, but must be left blank to its right. This is to give the program the ability to calculate the energy of subcomponents of the system, such as the ability (hopefully supported in future versions) to do counterpoise calculations. Finally, version 0.2b now writes the *binding* energies to `EnergyEnsemble.py` rather than the absolute energies.

2 Improvements in 0.1b from 0.0b

Version 0.1b features local elevation, also known as metadynamics, proposed by Huber and coworkers in 1994 [2]. The feature uses memory to penalize regions that have already been sampled. The feature is turned on by default in 1.1b, and may affect how you perform data analysis. The local elevation feature can be completely controlled by the variables `MetaDynamicsOn`, `MetaWidths`, and `MetaHeight`. See Section 10.3 for details on how to choose these variables. In future versions I hope to include more details on how to choose these features optimally.

3 Support

Please send questions, bugs, and requests to Tristan Truttmann at tristan.truttmann@gmail.com.

4 Purpose

The MC code was developed as a tool for modeling the physical interaction between energetic compounds and organic adsorbents; however, interaction between any adsorbate and adsorbent can be modeled[1]. The user is required to generate the initial structures of the adsorbent and admolecule, and then provide appropriate information to the SIESTA quantum chemistry package, as well as the appropriate pseudopotential files. The user is also required to provide simulation information into a preferences file such as the temperature and the size (in atoms) of the ad-molecule. The program will output the geometries and binding energies of every accepted structure.

5 Dependencies

The code was written for Python 2.6/2.7 and assumes that the standard Python libraries are installed as well as the `numpy` and `matplotlib` python libraries. It also assumes that it is running in an environment that has SIESTA 4.0 installed that is callable with the `siesta` command. If the user wishes to use the automatic admolecule attachment feature (where the user supplied two `.xyz` files rather than one), the user must have the python package `statsmodels` installed. This can be quickly remedied by downloading and installing the Anaconda data science platform [1].

6 Suggested use

The code was designed to be lightweight so that the user can copy the entire software directory to a desired parent directory, and then make the necessary changes to the `Siestafiles/` directory and the `MC_Pref.py` file. The simulation can then be run by either running `RunMC.py` directly (i.e. `./RunMC.py`), or by submitting `RunMC.py` to a job manager (i.e. `qsub RunMC.py`). It should be noted that, in order for the program to execute properly, the current working directory at the time of submission must be the directory that contains the `RunMC.py` file. It should also be noted that the first couple lines of `RunMC.py` contain PBS directives that should be modified to the user's liking (if she is using a job manager). Job managers that are not based on PBS are not supported, but if the user wants to, she may replace the PBS directories with those of her desired job manager. However, if she does this, she should ensure that the `TMPDIR` environment variable is set to a scratch path; this prevents excessive writing overhead to the user's home directory (the program will write temporary files to the directory that the job was submitted in if `TMPDIR` is not defined). The program will write all output files to the `output/` directory. If the user runs the simulation when there are already files in the output directory, the program will move all those files to a directory named `archive_output###` were `###` is an index number. This allows the user to resubmit simulations but keep track of previous program settings and results.

7 Getting Started

To ensure that your machine has the necessary prerequisites, submit the default job as a test using the following instructions. The default job simulates TNT interacting with α-cellulose.

[1] There are of course exceptions to this. For example, if your adsorbent or admolecule has less than 3 atoms, the program will run into trouble.

- Copy the software directory to any desired location on your machine. (e.g. `cp monte-calo-master /jobs/`)
- Move to the directory you just created. (e.g. `cd  /jobs/monte-carlo-master`)
- Submit the job if you have a batch manager (i.e. `qsub RunMC`) or run the script directly (i.e. `./RunMC.py`).
- After the job is done running, move to the output directory. Look at the output files. (e.g. `cd output && ls`)
- Look through your results. You can see the geometries by opening `GeomEnsemble.xyz` in some molecular viewing program such as Avogadro.

8 Function

This sections explains, intuitively, how the program works. The program aims to model the interaction between a *single* ad-molecule and an adsorbent surface. Further, it does this in a vacuum.

To estimate the binding energy and properly account for thermal energy in the system, a Metropolis Monte Carlo technique is implemented. This involves a random translation and rotation of the ad-molecule followed and preceded by a potential energy calculation using SIESTA. The random move is then accepted or rejected based on the standard Metropolis Monte Carlo acceptance criterion; this is:

$$W = \min\left[1, \exp\left(\frac{-\Delta E}{kT}\right)\right]$$

Here, W is the probability of accepting the move, ΔE is the change in potential energy during the move, k is the Boltzmann constant, and T is the temperature in kelvin. If the move is rejected, the geometry prior to the move is assumed and that geometry is again recorded. If the move is accepted, the new geometry is recorded, and another random move starting from the new geometry is taken.

The choice of random moves is important for program performance. Generally speaking, a move that has an acceptance rate of 50% is ideal. For this program, each move has both a translation component, and a rotation component. During the translation component, a random direction, uniformly distributed over the surface of a sphere, is chosen. Then a random translation distance is chosen from a flat distribution between zero and the value of `TransLimit`. For the rotation component, an axis of rotation is chosen randomly from a uniform distribution over the surface of a sphere. Then a random rotation angle is chosen from a uniform distribution between zero and the value of `RhoLimitDeg`. The initial values of `TransLimit` and `RhoLimitDeg` are imported from the `MC_Pref.py` file. If the file does not define these variables, then defaults are imported from `MC_Defaults.py`. If the user chooses `Optimize = False`, then these values are kept constant over the simulation. However, if the user chooses `Optimize = True`, then the these two variables are changed dynamically during the program to reach an acceptance rate near 50%. To do this, every accepted move is followed by increasing the values of both `TransLimit` and `RhoLimitDeg` by 5%, and every rejected move is followed by a decrease in these variables by 5%. The program then continues this process until there are `EnsembleTartget` accepted structures.

8.1 Metadynamics (a.k.a. local elevation)

If the user sets the variable `MetadynamicsOn = True`, then the algorithm above is slightly modified. After each move, the accepted geometry (or the geometry before an unaccepted move) has a "penalty bias" assigned to it. That is, there is an added potential:

$$E_{i,\text{penalty}}(x,y,z,\alpha,\beta,\gamma) = \epsilon e^{\frac{-1}{2}\left[\left(\frac{x-x_i}{\sigma_x}\right)^2+\left(\frac{y-y_i}{\sigma_y}\right)^2+\left(\frac{z-z_i}{\sigma_z}\right)^2+\left(\frac{\alpha-\alpha_i}{\sigma_\alpha}\right)^2+\left(\frac{\beta-\beta_i}{\sigma_\beta}\right)^2+\left(\frac{\gamma-\gamma_i}{\sigma_\gamma}\right)^2\right]}$$

Here, x, y, and z describe the position of the centroid of the admolecule, while α, β, and γ describe the orientation of the admolecule (proper Euler angles). The variables x_i, y_i, z_i, α_i, β_i, and γ_i represent the past position of the adsorbent after move i. The parameter ϵ represents the height of the penalty bias, and is picked by the user with the variable `MetaHeight` in the `MC_Pref.py` file. The paramters σ_x, σ_y, σ_z, σ_α, σ_β, and σ_γ are chosen by the user with the `MetaWidths` variable in the `MC_Pref.py` file. The variable `MetaWidths` is a 6-element numpy array, with the first element representing σ_x and the last element representing σ_γ.

After several iterations with local elevation turned on, the effective energy becomes:

$$E_{\text{eff}}(X) = E_{\text{SIESTA}}(X) + \sum_i E_{i,\text{penalty}}(X)$$

Where X describes all the variables x, y, z, α, β, and γ collectively. The new acceptance criterion then becomes:

$$W = \min\left[1, \exp\left(\frac{-\Delta E_{\text{eff}}}{kT}\right)\right]$$

It is recommended that *if* the user chooses `MetadynamicsOn = True`, then she should also choose to keep the maximum move size fixed i.e. `Optimize = False`. If both these options are set to `True`, then a simulation often continually decreases its maximum move size until it gets stuck in a penalized region. This is undesirable, of course. Finally, it is important to realize that turning on local elevation *will no longer produce structures according to the Boltzmann distribution.*

9 Files

This section explains the purpose of each file in the MC software. In the software directory, there are, by default, four files and two subdirectories. During operation, there may be an additional three files and several additional directories. These files and directories are summarized in Table 1. The files inside the subdirectories are summarized in Table 2 and Table 3.

10 Changing simulation preferences

To submit a custom job, the user is required to do four things:

- Supply an initial geometry to replace the `SiestaFiles/InputGeom.xyz` file.
- Edit the `SiestaFiles/template.fdf` file to her needs.
- Supply necessary pseudopotential files as `*.psf` where "`*`" are the symbols of each element.

Table 1: Summary of the files and subdirectories in the software directory.

`MC_Defaults.py`	Contains default preferences that are not defined in the `MC_Pref.py` file.
`MC_Defaults.pyc`	A file that python may generate automatically.
`MC_Pref.py`	A python file that the user must edit to change preferences of the simulation.
`MC_Pref.pyc`	A file that python may generate automatically.
`MC_Util.py`	The file that contains utilities for the `RunMC.py` file.
`MC_Util.pyc`	A file that python may generate automatically.
`RunMC.py`	The file that contains the core of the program as well as job management information. The program is called with this script (i.e. `./RunMC.py`)
`SiestaFiles/`	A folder that contains necessary information for SIESTA to run.
`archive_output###/`	"`###`" can be any number. These directories are generated to store output files and preferences from previous runs.
`output/`	All of the output output files are stored in this directory.

Table 2: Summary of the files in the `SiestaFiles` subdirectory.

`InputGeom.xyz`	A text file in xyz format that stores the initial geometry of the system being studied. The user must replace this with her system.
`adsorbent.xyz`	A file that contains the geometry of the adsorbent. This file is not included by default. If this file is present, then the file `admolecule.xyz` should also be present and the file `InputGeom.xyz` should be omitted. Also note that adsorbents must be roughly flat and thin for this method to work.
`admolecule.xyz`	A file that contains the geometry of the admolecule. This file is not included by default. If this file is present, then the file `adsorbent.xyz` should also be present and the file `InputGeom.xyz` should be omitted.
`template.fdf`	The input file for SIESTA, excluding the geometry block at the end (which the program will append automatically). The user must edit this file to her specific needs.
`*.psf`	"`*`" are chemical species labels defined in the `ChemicalSpeciesLabel` block of the`SiestaFiels/template.fdf` file. The pseudopotential files for SIESTA. The users must either generate these files or download them from the internet.

Table 3: Summary of the files in the `output` subdirectory.

`ConfigEnsemble.py`	A Python-readable file that stores all the information about every accepted MC structure in the system. This includes energy, geometry, lattice information, and the bias from metadynamics or umbrella sampling.
`EnergyEnsemble.py`	A Python-readable file that stores the binding energy values at every step in a numpy array.
`GeomEnsemble.xyz`	An xyz file that stores the geometries of every accepted structure.
`MC_Pref[archive].py/`	A copy of the `MC_Pref.py` file that was used for the job.
`SiestaFiles[archive]/`	A copy of the `SiestaFiles/` directory that was used for the job.
`time.log`	A file that records important performance information such as parallel speedup, acceptance ratio, and time spent in different parts of the program.
`output.pbs`	Only exists if submitted as batch jobs. A file that contains the standard output of the entire job.
`error.pbs`	Only exists if submitted as batch jobs. A file that contains the standard error out of the entire job.

- Edit simulation preferences in the `MC_Pref.py` file.
- If the user is submitting a batch job, edit the PBS directives at the beginning of `RunMC.py`. Then run `qsub RunMC.py`.
- If the user is submitting a regular script, run the command `./RunMC.py`.

Further details are provided in the following subsections:

10.1 Edit `SiestaFiles/template.fdf`

The only difference between the `SiestaFiles/template.fdf` file and a traditional SIESTA input file is that the former is missing the geometry block. Otherwise, all SIESTA options can be used except options that are not supported by this MC program such as:

- Molecular dunamics are not supported.
- TransSiesta calculations are not supported.
- Minimizations are currently not supported but may be in the future (see Section 11).
- Only the Harris functional is supported, but more functionals will be supported in the future (See Section **??**).
- The field `NumberOfAtoms` should be *included* but then *left blank* to the right. This is a temporary fix that may be improved in future versions.
- Only one pseudopotential file can be used per element. It is sometimes common for SIESTA users to use different pseudopotential files for the the same element in different environments, but this program does not support that.

Detailed information about the SEISTA input file syntax and options can be found at **their website**. Remember to change the number of atoms and the lattice information.

10.2 Pseudopotential files

Every element in your simulation must have a pseudopotential file. For common functionals, pseudopotentials can often be found in **pseudopotential databases**. For more "exotic" functionals like vdW-DFT, you may have to generate your own pseudopotentials using **the built-in psuedopotential genera** or a third party pseudopotential generator like **OPIUM**.

10.3 Edit `MC_Pref.py`

The `MC_Pref.py` file is in Python readable format, so any tricks or shortcuts you know in Python can be used to generate these variables. However, if you spell a variable incorrectly, the program will not read the option and use the program defaults. Therefore, *it is very important that you spell the option variables correctly.* For every system, the following required options must be changed:

- `AdsorbateSize`: This variable should be set to the number of atoms in your adsorbate
- `SubLatticeGrid`: If periodic boundary conditions are specified in the `SiestaFiles/template.fdf` file and `ShepherdOn = True`, then this variable should be defined. This is necessary to keep your ad-molecule confined to the smallest possible repeating unit of the adsorbent surface. So if your adsorbent used in you system includes 2 repeating unites in the first vector specified in `SiestaFiles/template.fdf`, and 3 repeating unites included in the second vector, then you should define `SubLatticeGrid = np.array((2,3,1))`. Of course, if this confuses you, you can always remove the variable, and the defaults will just limit the system to the unit cell defined in `SiestaFiles/template.fdf`.

The following options are more ways to tune the calculation to your needs:

- `T_Sequence`: This variable represents the temperature of the system, or the simulated annealing sequence of temperatures. If you want to run the calculation at a constant temperature, for example 298 K, use `T_Sequence = [[1,298]]`. However, if you want to run a sequence, use a nested list, where each element is a 2-list where the first value is the number of moves attempted at the temperature specified in the second subelement. The sequence will repeat itself after it is finished. For example, if `T_Sequence = [[1000,700],[1000,298]]`, then the simulation will make 1000 attempts at 700 K, and then 1000 attempts at 298 K, and then repeat the process until the program is complete.
- `EnsembleTarget`: This specifies the number of accepted structures that must be reached before the program terminates.
- `Optimize`: This logical variable, when set to True, will optimize the MC step size to approach an acceptance rate of approximately 50%.
- `TransLimit`: This variable is the maximum translation, in Angstrom, that the ad-molecule will be moved. If `Optimize = True`, then this value will be automatically adjusted during the program.

- `RhoLimitDeg`: This variable is the maximum rotation, in degrees, that the ad-molecules will be rotated during a MC step. If `Optimize = True`, then this value will be automatically adjusted during the program.
- `ShepherdOn`: This logical variable, if set to `True`, will prevent the ad-molecule from walking out of the unit cell specified by the lattice info specified in `SiestaFiles/template.fdf` and `SubLatticeGrid`.
- `NumberOfSiestaCores`: This variable specifies the total number of CPU cores to run SIESTA on.
- `MetaDynamicsOn`: This logical variable, when set to True, turns on local elevation. See Huber and coworkers for more specific details on the algorithm [2].
- `MetaWidths`: This variable is a six-element numpy array that specifies the standard deviation widths of the penalty function. The first three elements are the x, y, and z Gaussian widths (the directions as specified in the `InputGeom.xyz` file) in ngstrm and the last three elements are the Gaussian withs for the proper Euler angles α, β, and γ in degrees. The three Euler angles descript the orientation of the admolecule and are generated internally in the program. If you are particularly interested in how these angles are defined, see the source code in file `M_Util.py` near line 509 for details.
- `RightSide`: This optional variable is a 3-element numpy array that points in the direction of the side of the adsorbent that you would like to attach the admolecule to. This option is only used if the user supplied the two geometry files `adsorbent.xyz` and `admolecule.xyz`. If the user does not care which side the admolecule is attached to, then she may set this variable to `None`.
- `CounterPoiseOn`: This option is not supported yet and must be set to `False`. In future versions, this may provide the option of automatically doing counterpoise calculations.

11 Coming features

During the fall of 2016, I will be working to implement the following features as an undergraduate honors project. I hope to have all improvements implemented by December 2016:

- **Covalent interaction relaxation**: The current program models the ad-molecule and adsorbent as rigid independent bodies. In coming versions, I will supply the option to relax the geometry of each of these independent bodies at every MC step before calculating the energy used in the acceptance criteria.
- **Basis set superposition error**: Noncovalent binding energy calculations are known to have a large error due to basis set superposition error. I hope to provide an option to correct for this in the future. I am currently facing difficulties from SIESTA in implementing this option.
- **Improved parallelism**: The current version supports all parallelism that is supported in SIESTA. However, I may add additional embarrassing parallelism to run on tens to hundreds of nodes if necessary.

If you have requests for additional features, or are waiting for one of the features listed above, please email Tristan Truttmann at tristan.truttmann@gmail.com, so I can be honest with whether I can implement the requested features and on what time frame.

12 License

13 Acknowledgements

This work was supported by the NSF DMR under award no. 1359239 SusChEM: REU Site: Polymer Innovation for a Sustainable Future. I would also like to thank several individuals who have given me assistance during the project including Gopinath Subramanian, Guido Todde, Sanjiv Jha, and Michael Roth.

References

[1] Continuum Analystics. Anaconda home page.

[2] Thomas Huber, Andrew E. Torda, and Wilfred F. van Gunsteren. Local elevation: A method for improving the searching properties of molecular dynamics simulation. Journal of Computer-Aided Molecular Design, 8(6):695–708, 1994.