Efficient adsorption studies with DL_MONTE

Tom L. Underwood, Tina Düren, John A. Purton, Nigel B. Wilding

8 March 2019

Abstract

DL_MONTE is a general-purpose simulation program for applying the Monte Carlo method to condensed matter. Here, we provide an overview of our recent project to significantly improve the capabilities of DL_MONTE for studying adsorption.

DL_MONTE is accompanied by a Python toolkit for managing simulation workflows and performing data analysis. Improving this toolkit was the focus of the project. We developed Python functions to facilitate analysis of time series output by DL_MONTE (and other molecular simulation programs): we developed functions to calculate uncertainties using block averaging, and to calculate the correlation and equilibration times of time series. Moreover, we developed functions to apply the multiple histogram reweighting method. Building on these functions, we developed a Python framework for automating the task of calculating a specified physical quantity to a desired precision using DL_MONTE. The framework can be used to automate the process of calculating an adsorption isotherm with DL_MONTE, and is sufficiently general that it could be easily adapted to treat other simulation programs.

Finally, we extended the grand-canonical Monte Carlo functionaity of the DL_MONTE main program to include free energy calculations. A key application of this new functionality is pinpointing the location of liquid-gas phase transitions, something which was hitherto intractable with DL_MONTE except in special cases.

1 Introduction

Adsorption is the process whereby a substance (the adsorbate), usually a fluid, forms a thin film on an external or internal surface of a material. Adsoprtion underpins many key fields of research, including catalysis, carbon capture and storage, and surface and interfacial phenomena. There is therefore a demand for software which can be used to study adsorption efficiently, i.e. with minimal computational cost.

Grand-canonical Monte Carlo (GCMC) [1] is a molecular simulation method commonly used to study adsorption. GCMC samples the equilibrium states of the

system at a specified temperature, system volume, and chemical potential (of the adsorbate). The key feature of GCMC is that particles can be added and removed from the system during the course of the simulation. This affords it considerable advantages over other molecular simulation methods, including molecular dynamics, for studying adsoption.

DL_MONTE [2,3] is a general-purpose Monte Carlo program which can be used to study adsorption in a wide range of systems using GCMC, including those of relevance to the fields given above. DL_MONTE is part of the suite of simulation software developed by Daresbury Laboratory and the Collaborative Computational Project 5 (CCP5), and is accompanied by a Python toolkit for managing DL_MONTE simulation workflows and performing data analysis.

We have recently undertaken an ARCHER embedded CSE programme project whose aim was to significantly improve the capabilities of DL_MONTE for studying adsorption. This entailed adding new functionality to both the Python toolkitand the main DL_MONTE program. In this report we provide an overview of this new functionality.

2 Python framework for automating simulations

A common task for molecular simulation is to calculate the value of a certain physical quantity *X* (e.g. the density of the system) to a given precision. This typically entails the following:

1. Run a simulation of a prescribed length.
2. Extract a value and uncertainty for *X* using the time series for *X* output by the simulation.
3. Check whether or not the uncertainty in *X* is less than the desired precision. If it is, then the task is complete. If it is not, then return to step 1: perform another simulation, starting where the previous one left of, extending the time series for *X*.

We have developed Python tools to automate this process. While the focus was on automating the process of calculating adsorption isotherms (i.e. calculating the density of adsorbate vs. chemical potential, at a given temperature) using DL_MONTE, the framework is sufficiently generalised that it could be applied to other problems and simulation codes. We provide an overview of these tools below.

2.1 Time series analysis

We first developed Python functions to facilitate step 2 in the above procedure. These were drawn upon when developing the Python framework for automating the above procedure, which is discussed in the next subsection.

The key output of molecular simulation programs is a time series of physical quantities. To extract a value and an associated uncertainty for the physical quantity of interest, *X*, one must analyse such time series. Analysis is complicated by the fact that data from the start of a simulation must often be disregarded,

because it pertains to times before the simulation has equilibrated. Thus a calculation of X from the simulation data entails averaging over all values of X in the time series after the *equilibration time*.

Calculating the statistical uncertainty in X is a less straightforward task. The issue is that nearby values of X in the time series are typically highly correlated, and standard formulae for estimating uncertainties rely on the observations pooled to calculate the uncertainty being uncorrelated. *Block averaging* [1] is a standard method which addresses this problem. In block averaging, one partitions the time series – after the equilibration time – into blocks whose lengths, the *block size*, are significantly larger than the correlation time in the time series. The mean value of X is then evaluated for each block. The mean values for all blocks are then used to evaluate the standard error of the mean in the conventional manner – which is valid because the block means should be uncorrelated. Of course, the validity of this method hinges upon an appropriate choice of block size. One must know the correlation time for the time series in order to inform this choice.

We have developed Python functions which can be used to:

i. determine whether or not a given time series has reached equilibrium
ii. provide an upper bound for the equilibration time
iii. calculate the correlation time of a time series
iv. calculate an uncertainty for the physical quantity the time series pertains to using the block averaging method.

These functions make it significantly easier to calculate uncertainties from the output of molecular simulation programs, and to embed this ability within Python scripts and modules.

2.2 Framework for automation

Building on the analysis functions just described, we have developed a Python framework which automates the process of calculating a specified physical quantity and its uncertainty. The framework allows the user to specify a physical quantity to be evaluated, and either a precision to which it is to be evaluated to or a maximum wall-clock time to dedicate to the calculation. It will then perform back-to-back DL_MONTE simulations until enough data is gathered such that the physical quantity is determined to the desired precision or the maximum wall-clock time has elapsed. The framework also allows this task to be repeated for a range of thermodynamic parameters (e.g. temperature, pressure), as demonstrated below. Moreover, the abstraction in the framework is such that it could easily be applied to simulation programs other than DL_MONTE.

The key ingredients of the framework are Python classes Task and TaskInterface. The former corresponds to a simulation workflow to achieve a particular aim. Subclasses of this include Measurement, which calculates a physical quantity and its uncertainty at a given thermodynamic parameters, say, temperature and chemical potential; and MeasurementSweep, which applies Measurement to a specified range of thermodynamic parameters. Each Task instance has a

TaskInterface attribute which defines how to perform common simulation tasks, such as run a simulation, alter input variables, and extract data, and is specific to the simulation program, e.g. DL_MONTE; for each program, a TaskInterface interface must be written which tells Task instances how to perform the common tasks. Calling the run() function of a Task instances instigates the automated workflow corresponding to the Task.

2.3 Example: Automated calculation of an isotherm

Figure 1 shows the output of the framework when tasked to evaluate the density ρ vs. chemical potential μ for the k_BT/ε=1.5 isotherm of the Lennard-Jones fluid. Each point in the figure corresponds to a set of GCMC simulations performed back-to-back until a maximum wall clock time is exceeded. The salient features of the Python code used in this case can be found in the following code snippet:

```
# 'simtask' is the package housing the functionality
import simtask.task as task
import simtask.measurement as measurement

# Link the framework to a DL_MONTE executable
interface = task.interface.DLMonteInterface("DLMONTE-SRL.X")

# Choose to calculate the number of molecules in the system
nmol_obs = task.Observable( ("nmol",0) )
observables = [ nmol_obs ]

# Define what constitutes a single 'measurement'
measurement_template = measurement.Measurement(interface, observables, maxsims=20,
maxtime=30)

# Define the nature of multiple measurements: here
# we perform measurements at the various 'molchempot'
# in the list 'chempots'
sweep = measurement.MeasurementSweep(param="molchempot",
        paramvalues=chempots,
        measurement_template=measurement_template)

# Instigate the workflow
sweep.run()
```

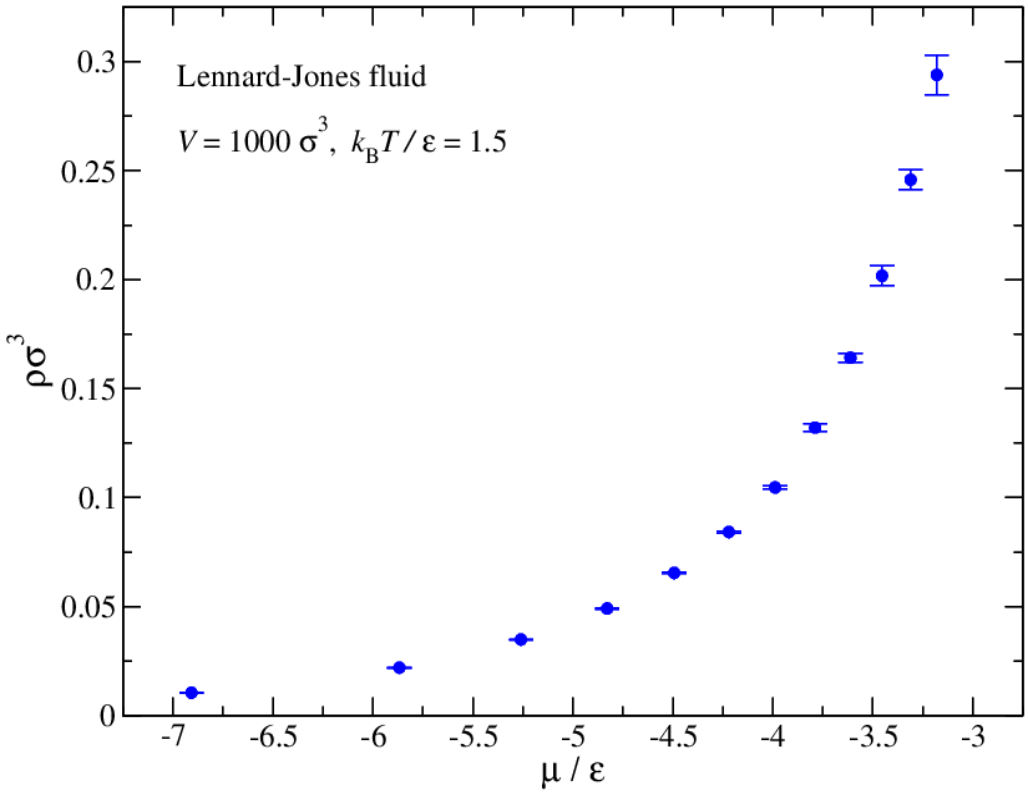

Figure 1: Isotherm for the Lennard-Jones fluid automatically generated by the Python framework.

3 Multiple histogram reweighting

Histogram reweighting [4,5] is a powerful method which allows simulation data obtained at certain thermodynamic parameters (e.g. temperature, pressure) to be used to make predictions at thermodynamic parameters not explored by simulation. *Single histogram reweighting* [4] is the simplest incarnation of the method, and involves using data obtained from a *single* simulation.
This method was already implemented in the DL_MONTE Python toolkit – an output of a previous eCSE project [6]. In this project we have extended the toolkit to be able to perform *multiple histogram reweighting* (MHR) [5], in which data from *multiple* simulations can be used.

3.1 Methodology

Here we describe MHR within the grand-canonical thermodynamic ensemble, where the temperature *T,* volume *V*, and chemical potential μ of the (adsorbate) particles μ are fixed. This is the ensemble sampled in GCMC simulations. Note, however, that MHR is a general method which can be applied to a wide range of thermodynamic ensembles, and so also can our MHR software, as discussed below.

Suppose we have performed R GCMC simulations at various chemical potentials and temperatures, $\mu_1, \mu_2, \ldots, \mu_R$, and $T_1, T_2, \ldots, T_R$. MHR enables us to use the data from all of these simulations to predict the value of a certain physical quantity X at a temperature T' and chemical potential μ' not treated by any of the simulations. The relevant equations are as follows:

$$\langle X' \rangle = \frac{\sum_{n=1}^{R} \sum_{i=1}^{D_n} X_{ni} \exp\left[(-\beta' + \beta_n)E_{ni} + (\beta'\mu' - \beta_n\mu_n)N_{ni} - F_n\right]}{\sum_{n=1}^{R} \sum_{i=1}^{D_n} \exp\left[(-\beta' + \beta_n)E_{ni} + (\beta'\mu' - \beta_n\mu_n)N_{ni} - F_n\right]}$$

where $\beta=1/(k_BT)$; $\beta_n=1/(k_BT_n)$; X_{ni}, N_{ni}, and E_{ni}, are the values of X, the energy and the number of particles for the ith configuration in the nth simulation; D_n is the number of configurations sampled in the nth simulation; and $F_1, F_2, \ldots, F_R$ are given by

$$e^{-F_n} = \sum_{n=1}^{R} \sum_{i=1}^{D_n} \left\{ \sum_{m=1}^{R} D_m \exp\left[(-\beta_m + \beta_n)E_{ni} + (\beta_m\mu_m - \beta_n\mu_n)N_{ni} + F_m\right] \right\}^{-1}$$

which must be solved using an iterative procedure.

3.2 Generality

Our implementation of MHR is sufficiently generalised that it can treat a wide range of ensembles. To elaborate, a MHR function is provided for applying to thermodynamic ensembles with the general form

$$p(\mathbf{X}_i) \propto \exp(\mathbf{b} \cdot \mathbf{X}_i)$$

where $p(\mathbf{X}_i)$ is the probability of a configuration i with a vector of physical quantities $\mathbf{X}_i$ coupled to a vector of thermodynamic parameters $\mathbf{b}$. For instance,

$$\mathbf{b} = \left(-1/(k_BT), \mu/(k_BT)\right)$$

$$\mathbf{X}_i = (E_i, N_i)$$

in the grand-canonical ensemble

where E_i and N_i denote the energy and number of particles for configuration$i$. However, for the convenience of users MHR functions are provided for the grand-canonical, canonical (fixed number of particles N, V and T), and isothermal-isobaric (fixed N, pressure, T) ensembles which take familiar quantities such as E_i and N_i as arguments – to save users from having to 'translate' into the generalised thermodynamic ensemble just described.

3.3 Example: Interpolation of an isotherm

Figure 2 shows the results of applying MHR to interpolate the $k_B T/\varepsilon$=1.5 isotherm of the Lennard-Jones fluid, using GCMC simulation data obtained at three chemical potentials. The salient features of the Python code used in this case can be found in the following code snippet:

```
import multihistogram
import numpy

# The three GCMC simulations each output a time series
# of E and N. These have been imported and stored in
# 'E_data' and 'N_data', which are both lists containing
# the 3 E and N time series as arrays.
# 'sysvol' and 'kT' are the same for all simulations.
# 'mu_sims' contains the chemical potentials used in the
# simulations

kT_sims = numpy.ones(len(mu_sims)) * kT

# We are interested in the density; calculate the time
# series
rho_data = N_data / sysvolume

# 'mu_for_mhr' is a list of chemical potentials to
# reweight to. Print the density calculated using MHR
# at each chemical potential
for mu in mu_for_mhr:

    rho_mhr = multihistogram.reweight_observable_muvt(kT_sims,
            mu_sims, E_data, N_data, obs=rhodata,
            kT_new=kT, mu_new=mu)

    print(mu, rho_mhr)
```

3.4 Limitations

The limitations to MHR should be emphasised. The method relies on, for the case of the grand-canonical ensemble, an overlap in the range of E and N sampled by the simulations. In practice this means that MHR is limited to making predictions at T' and μ' 'close' to those of the simulations.

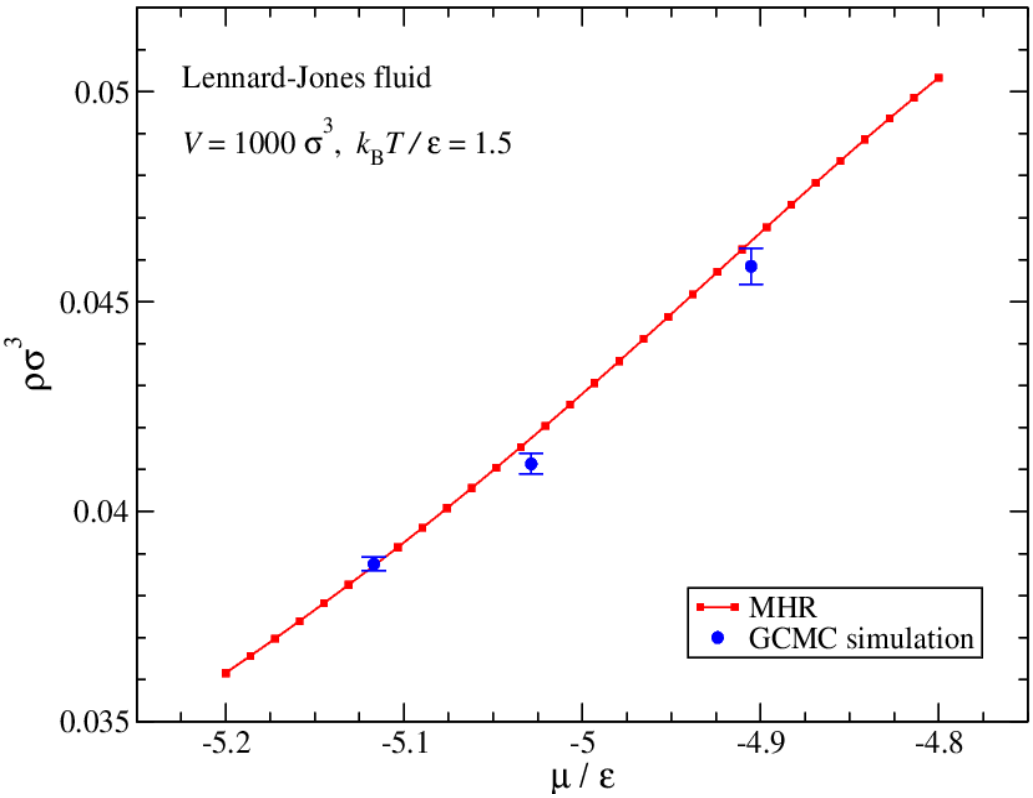

Figure 2: Isotherm for the Lennard-Jones fluid obtained by applying MHR to DL_MONTE GCMC simulation data.

4 Free energy methods with grand-canonical Monte Carlo

Liquid-gas phase transitions play an important role in adsorption phenomena, and there is much interest in using molecular simulation to pinpointing the location of such transitions. Solving this problem entails calculating the probability distribution for the density of the fluid, ρ, at various fluid chemical potentials μ. Close to the transition the probability distribution, $p(\rho)$, exhibits two local maxima, one at a low density ρ_G which corresponds to the gas phase, and one at a high density ρ_L which corresponds to the liquid phase. Pinpointing the transition amounts to finding the μ at which both peaks are of equal height. Unfortunately, in the vicinity of a phase transition calculating $p(\rho)$ is challenging. The issue is that between the high-probability regions in $p(\rho)$ centred on ρ_G and ρ_L is a region of densities with extremely low probability. Hence moving from ρ_G and ρ_L and back again – something necessary to measure $p(\rho)$ – is typically an event of exceedingly low probability that occurs only on timescales inaccessible to standard simulation methods.

Free energy methods address such problems by adding a fictitious contribution to the energy of the system, a *bias,* which encourages movement between two regions of configuration space connected by a region of low probability. The appropriate bias is usually not known from the outset, and is evolved during the course of the simulation until quick movement to and from the two regions is

eventually realised. At this point the effects of the bias can be removed from the simulation data using post-processing, and the equilibrium phase can be determined. (See [2] for an overview of free energy methods in the context of DL_MONTE).

4.1 New functionality

Free energy methods have been available in DL_MONTE for some time [2], however they could not be used in conjunction with GCMC. We have extended the existing free energy functionality in DL_MONTE to GCMC; GCMC can now be used in conjunction with the many free energy methods supported in DL_MONTE, including umbrella sampling, multicanonical Monte Carlo, Wang-Landau, and transition-matrix Monte Carlo [2]. As alluded to above, a key application of this new functionality is pinpointing the location of liquid-gas phase transitions.

4.2 Example: Liquid-gas coexistence in methane

To test this new functionality we used transition-matrix Monte Carlo (TMMC) with GCMC, in conjunction with histogram reweighting, in order to pinpoint the liquid-gas transition for methane (modelled using the TraPPE-EH force field [7]) at 150K. Figure 3 shows the free energy vs. density at various chemical potentials μ for this system. The free energy for density ρ here is defined as $F(\rho)$=-ln $p(\rho)$ (up to an arbitrary additive constant), where $p(\rho)$ is the probability of the system exhibiting density ρ. Low $F(\rho)$ correspond to high $p(\rho)$. Each free energy curve in the figure was obtained by applying histogram reweighting to data from a single TMMC+GCMC simulation. Note that $F(\rho)$ exhibits two local minima, which correspond to ρ_G and ρ_L. At the phase transition the gas and liquid probabilities are equal, and hence $F(\rho_G)$ and $F(\rho_L)$ are roughly equal. The μ corresponding to the phase transition was determined by using reweighting as described in [8]. Note that the ρ_G and ρ_L at the transition are in excellent agreement with the results in the literature [7].

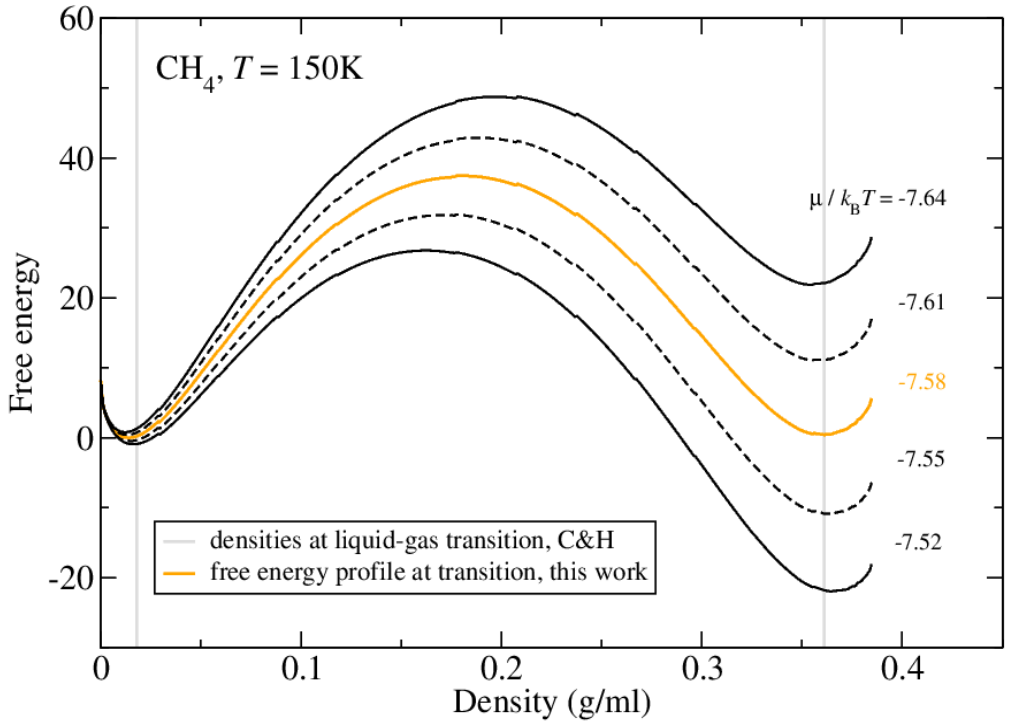

Figure 3: Free energy profiles obtained by GCMC free energy method functionality in DL_MONTE, and histogram reweighting, for methane modelled with the TraPPE-EH force field. 'C&H' signifies Chen & Siepmann [7].

5 Conclusions

This project entailed adding new functionality to both the DL_MONTE Python toolkit and the DL_MONTE main program. With regards to the toolkit, we developed Python functions to facilitate analysis of time series output by molecular simulation programs. Building on these functions, we developed a Python framework for automating the task of calculating a specified physical quantity to a desired precision. One application of the framework is to automate the process of calculating an adsorption isotherm with DL_MONTE. Moreover, it should be noted that the framework is sufficiently general that it could be easily adapted to treat other simulation programs. These new Python tools are beneficial because they reduce the 'actual time' taken to solve a given problem, where by 'actual time' we mean the total time taken for the necessary workflow of simulations to complete, in addition to the time the user spends preparing input files, writing their own analysis software, analysing data between simulations, etc.

We also developed Python functions for applying the multiple histogram reweighting (MHR) method. MHR allows one to make the most of the data one has already generated, reducing the computational cost associated with determining how a physical quantity (e.g. density of adsorbate) varies over a range of thermodynamic conditions.

The improvement we made to the main DL_MONTE program was to extend the grand-canonical Monte Carlo (GCMC) capabilities of DL_MONTE to include free energy calculations. A key application of this new functionality is pinpointing the location of liquid-gas phase transitions, something which was hitherto intractable with DL_MONTE except in special cases. Keeping in mind that DL_MONTE's flexible force field enables it to treat a wide variety of systems, this new functionality will enable the liquid-gas transition to be studied with DL_MONTE in a plethora of systems. This would be useful for, e.g. the development of models of fluids, and in studying phenomena where the liquid-gas transition plays a crucial role, such as wetting and drying.

The DL_MONTE homepage [9] and GitLab project [10] describes how to obtain access to the main DL_MONTE program and the Python toolkit. The aforementioned improvements to the main DL_MONTE program are included in the latest release of DL_MONTE, v2.06. An initial release containing the Python tools developed during this project is imminent.

References

[1] See, e.g., D. Frenkel and B. Smit, 'Understanding molecular simulation: from algorithms to applications', San Diego: Academic Press (2002)
[2] A. V. Brukhno et al., Mol. Sim., DOI:10.1080/08927022.2019.1569760 (2019)
[3] J. A. Purton, J. C. Crabtree & S. C. Parker, Mol. Sim. 39, 14 (2013)
[4] A. M. Ferrenberg & R. H. Swendsen, Phys. Rev. Lett. 61, 2635 (1988)
[5] A. M. Ferrenberg & R. H. Swendsen, Phys. Rev. Lett. 63, 1195 (1989)
[6] eCSE04-4 'Implementing lattice-switch Monte Carlo in DL_MONTE to unlock efficient free energy calculations'
[7] B. Chen & J. I. Siepmann, J. Phys. Chem. B 103, 5370 (1999)
[8] N. B. Wilding, Am. J. Phys. 69, 1147 (2001)
[9] http://www.ccp5.ac.uk/DL_MONTE
[10] http://gitlab.com/dl_monte

Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk)