User manual for MDMC for pre-release v0.1 Nov 2017

# DISCLAIMER

This version of the software has only been tested with argon data and, therefore, this version is very much a pre-release version.

# Introduction

MDMC stands for Molecular Dynamics Monte Carlo.

This user manual is for the code and algorithm written and developed during parts of 2005-2008.

The software was created to test a new algorithm for fitting/optimising Potential Energy (PE) parameters against dynamical structure factor information. Although created with this specific purpose in mind the software was written so as to be extendable and expandable, see tech\_doc\_release\_v0.1.doc in folder doc.

At this point in time there is only support for fitting/optimising Lennard-Jones PE parameters.

For the source code see <https://github.com/MDMCproject/MDMC> and for the source code used for building v0.1 see <https://github.com/MDMCproject/MDMC/releases>.

# To run

MDMC takes one input, which is an MDMC *job file*.

To run MDMC:

1. Type mdmc.exe
2. When asked for a job file specify the full name of a job file

As a first test you may try the job file, md\_control\_argon.xml, located in the job\_files folder.

MDMC outputs are writing to the folder output (if it does not exist it will be created) where mdmc.exe is also located.

The folder for\_plotting\_results contains various Matlab files for plotting and investigating MDMC outputs; for more detail on this see a section further below.

# The Job file

So the job file is what controls the simulation. A job file will always contain this element:

<**control-object** name="mdmc\_control\_time\_corr">

This element has the attribute name which specifies the algorithm that MDMC will run. As of this writing MDMC includes these algorithms:

* mc\_control (just contain pseudo-code – in theory you could use the MDMC framework to run Monte Carlo (MC) only algorithms like Reverse MC)
* md\_control (run a Molecular Dynamics (MD) simulation and outputs Radial Distribution Function (RDF) information; also called the pair distribution function or the pair correlation function)
* md\_control\_time\_corr (same as above, although this go a step further by calculating space and time correlation information and from this e.g. calculates S(q, omega) data)
* md\_gridsearch\_control (calculate FOM over a grid of PE parameters against an RDF data set)
* mdmc\_control (does an optimisation of PE parameters against RDF data using Metropolis MC)
* mdmc\_control\_time\_corr (same as above but against dynamical structure factor data or other dynamical and structural data)

These algorithms were historically developed as iterative steps towards created mdmc\_control\_time\_corr.

The algorithm mdmc\_control\_time\_corr optimises MD parameters against dynamical and structural information; as of this writing this is supported for: S(q, omega), S(q, t) and ‘data’ in the form of the distinct part of the space-time pair correlation function gd(r,t).

The MDMC framework allows entirely new algorithms to be added in a pluggable way, although, this requires adding code and therefore see the technical documentation (tech\_doc\_release\_v0.1.doc in folder doc) for more on this.

This user manual focuses on writing a job file for the algorithm mdmc\_control\_time\_corr. Options available to the other algorithms are, with a few exceptions, a subset of the options available for mdmc\_control\_time\_corr.

The job file require a structure element:

<!-- Define structure from file -->

<!--

<**structure** filename="some\_structure.xml"> </structure>

-->

<!-- Alternatively build structure in code -->

<**structure** what-init-structure-to-build="fcc">

<title>Liquid Ar</title>

<density units="atom/AA3" val="0.0176" />

<number-of-unit-cells nx="6" ny ="6" nz="6" />

</structure>

The structure element is used to define the starting structure. This can be done in two ways as illustrated above, either by <structure filename="some\_structure.xml" or <structure what-init-structure-to-build="fcc". In the former case by naming a file which must conform to the file format:

<molecule title="rho = 0.02026">

<this-file-was-created when="Date 26/06/2017; time 16:02:33" />

<box-edges units="AA" x="34.93459" y="34.93459" z="34.93459" />

<atomArray number="864" units="AA">

<atom id="1" elementType="Ar" x3="12.304995" y3="-7.918292" z3="-12.436355" />

<atom id="2" elementType="Ar" x3="-15.329575" y3="-3.371399" z3="-17.431598" />

<atom id="3" elementType="Ar" x3="17.017586" y3="-5.852872" z3="15.047124" />

... etc.

<atom id="864" elementType="Ar" x3="6.445993" y3="12.690932" z3="4.848541" />

</atomArray>

</molecule>

As of this writing, for example, the algorithm md\_control writes out such a file at the end of its execution.

For the other case a starting structure is built in code from the information provided. The attribute what-init-structure-to-build accepts the values: “fcc” and “simple-cubic”. The structure created has the specified density (in units of atoms per cubic AA) with a box of nx, ny and nz unit cells in the x, y and z directions. The x-axis box edge length is therefore nx\*density^(-1/3) for a simple-cubic and 4^(1/3)\*nx\*density^(-1/3) for an fcc structure.

If specified, as shown below, the nearest neighbour method will be used in MD simulations. The implementation used is taken from the book by (Rapaport, 2004). The current implementation sets r-cut = 'smallest box edge'/2 - delta-r when r-cut+delta-r > 'smallest box edge'/2.

<**use-near-neighbour-method**>

<r-cut units="AA" val="7.491" />

<delta-r units="AA" val="1.364" />

</use-near-neighbour-method>

A number of least-squares Figure-of-Merits (FOMs) are supported, one for each type of input data:

<**fom**>

<!-- <g-d-rt-fom>

<data-file filename="data/argon/mdmc\_generated\_g\_d\_data.xml" />

<r-cut units="AA" min="0.0" max="10000.0" />

<t-cut units="AA" min="0.0" max="10000.0" />

<scale-factor val="1.0" />

<sigma val="1.0" />

</g-d-rt-fom>

<s-qt-fom>

<data-file filename="data/argon/mdmc\_generated\_s\_qt\_data.xml" />

<scale-factor val="1.0" />

</s-qt-fom> -->

<s-qo-fom>

<data-file filename="data/argon/Well\_s\_q\_omega\_Ag\_data.xml" />

<scale-factor val="1.0" />

<ignore-errors /> <!-- If data contains errors ignore these -->

</s-qo-fom>

</fom>

The scale-factor attribute is applied to the calculated data and defaults to 1.0. The FOMs that carry dynamical information are:

* <s-qo-fom>: S(Q, ω)
* <s-qt-fom>: S(Q, t)
* <g-d-rt-fom>: “normalised” distinct part of space-time pair correlation function gd(r,t)

The input data as specified by data-file is assumed to represent

S(q,omega) = 1/(2\*pi) \* integral ( exp(i\*[q\*r-omega\*t])\*G(r,t)\*dr\*dt )

Where G(r,t) is the space-time correlation function (also called van Hove correlation function and time-dependent correlation function). MDMC uses the units of AA for length and 10^-13s=0.1ps for time. And as of this writing the required units for S(q,omega) data for q is 1/AA, omega is 1/(0.1ps) and S(q,omega) is 0.1ps. Further

S(q,t) = integral ( exp(i\*q\*r)\*G(r,t)\*dr )

S(q,t) is dimensionless and the required unit for q is 1/AA and t is 0.1ps. Finally, introduce the distinct part of space-time pair correlation function gd(r,t)=Gd(r,t)/ ρ, where ρ = N/V i.e. the density. The “normalised” version is: ĝd(r,t)=N\* gd(r,t)/(N-1) which has the visual convenient property that it approaches 1 when t->∞ and r->∞. Not sure if data are/will ever be measured in form of gd(r,t), but this FOM may nevertheless be useful in some cases. gd(r,t) is dimensionless and r is in unit of AA and t in 0.1ps.

Also supported is the structural only data type <rdf-fom>; an RDF (also called pair distribution function and pair correlation function) based FOM:

<fom>

<rdf-fom>

<rdf-data filename="data/argon/mdmc\_generated\_rdf\_data.xml" />

<r-max units="AA" val="10.0" />

<scale-factor val="1.0" />

<sigma val="1.0" />

</rdf-fom>

</fom>

An RDF equals gd(r,t=0), is dimensionless and r is assumed to be in the unit of AA.

All input data files are required to use an XML format. These formats, one for each data type are also used by the software to save calculated data. The MDMC framework can be extended to accept other formats by adding a new reader file, see the technical documentation. Examples of each input data XML format supported can be found in the folder: data/argon.

A generalised Potential Energy (PE) function is specified using:

<**gpe**>

<lj-potential>

<sigma units="AA" val="4.0" fixed="no" min="1.0" max="6.0" max-move="0.2" />

<epsilon units="KJ/mole" val="1.5" fixed="no" min="0.2" max="5.0" max-move="0.2" />

<!-- you can argue whether r-cut below is a property of this potentially,

but for now you can optionally specify an r-cut value here, which

simply put the potential to zero for all r values higher than r-cut

-->

<r-cut units="AA" val="100.0" />

</lj-potential>

</gpe>

This is the PE function used in the MD simulations. At this point the only choice available is to specify a Lennard-Jones potential. Further choices can be added as explained in the technical documentation. In the example above the PE parameters are sigma and epsilon. The fixed attribute fixes a PE parameter if its value is different from “no”. Otherwise a parameter will be randomly moved according to the formula: max\_move\*(ran\_num-0.5) where ran\_num is a number between 0 and 1, although not below or above the specified min and max values.

## <control-object> in more detail

As stated earlier a job file requires a control-object element. It contains the user input options to control an algorithm. For the case where name="mdmc\_control\_time\_corr" is selected this means controlling the MD simulations and the MC Metropolis. See also the example job file mdmc\_control\_time\_corr\_argon.xml located in the folder job\_files.

To control the target temperature for the MD simulations and the MD time step (also referred to as a MD delta-t):

<temperature units="Kelvin" val="120.0" />

<time-step units="10e-13s" val="0.1075" />

To control the initial MD simulation:

<total-steps-initial-equilibration number="50000" average-over="1000" />

<perform-initial-temperature-calibration>

<total-step-temp-cali number="40000" />

<adjust-temp-at-interval number="2000" />

</perform-initial-temperature-calibration>

This MD simulation is done before starting the optimisation of the PE parameters. The starting phase-space configuration is that defined by the structure element with randomly assigned velocities (attempted to be) scaled to match the assigned temperature (for more details see the source code file phasespace.f90).

The elements total-steps-initial-equilibration and perform-initial-temperature-calibration and its attributes control this initial simulation. The former is used to set the total number of MD time steps and the latter how many of these to use for calibrating the temperature by scaling the velocities up or down at intervals determined by the number attribute of adjust-temp-at-interval. The average-over attribute sets the number of MD time steps to average over for calculating the kinetic energy, total energy etc. in order to reduce statistical noise. These averages are used for printing purposes and for judging if a simulation has reached an acceptable equilibrium; as of this writing, required to be within 20% of the target temperature and the total energy must drift less than 10% from the last temperature adjustment to the velocities.

The MC part of the algorithm is controlled by:

<mc-steps number="200" />

<temperature-mc val="1" />

mc-steps controls the number of MC steps (cycle) where new PE parameters are randomly selected according to that described above for the gpe element. Is used to control the Metropolis acceptance for these random moves: .

The MD simulations following the selection of PE parameters are controlled with:

<md-steps-repeated-equilibration number="50000" average-over="1000" />

<perform-repeated-temperature-calibration>

<total-step-temp-cali-repeated number="40000" />

<adjust-temp-at-interval-repeated number="1000" />

</perform-repeated-temperature-calibration>

This elements and attributes are names slightly different from those used for the initial MD simulation but otherwise should be self-explanatory.

How the histogram g(r,t) is calculated and how the time binning of g(r,t) is set is controlled by

<time-correlation>

<n-g-r-t-to-average-over number="20" />

<n-time-bin number="250" />

<md-per-time-bin number="7" />

<n-md-step-between-buffers number="100" />

</time-correlation>

n-g-r-t-to-average-over simply just tells how many times to calculate g(r,t) to reduce statistical noise. n-time-bin the number of time bins to use for g(r,t) and md-per-time-bin the width of a time bin measured in the of MD time steps. n-md-step-between-buffers is used to control how far apart in time the individual g(r,t) are calculated. The smaller this number the faster it takes to calculate n-g-r-t-to-average-over g(r,t)s, but picking it too small introduce unwanted correlation between the calculated g(r,t). In the book by Rapaport where the concept of using ‘buffers’ is taken writes “overlap (between buffers) should be confined to time intervals over which the correlation between measurements has decayed to a comparatively small value”. Note n-md-step-between-buffers must not be bigger than n-time-bin\*md-per-time-bin since this would slow don’t the calculation of g(r,t) compared to not using ‘buffers’. The number of buffers will approximately equals n-time-bin\*md-per-time-bin/n-md-step-between-buffers and this number must be less than or equal to n-g-r-t-to-average-over. In order to avoid the calculation of any partial buffer(s) the software may adjust n-time-bin slightly to make it equal a multiple of the spacing between buffers.

As of this writing the radial binning, r-binning, of g(r,t) is control by:

<calculate-rdf>

<!-- <cal-rdf-at-interval number="50" />

<average-over-this-many-rdf number="20" /> -->

<r-max units="AA" val="50.0" />

<bin-length units="AA" val="0.2" />

</calculate-rdf>

Where r-max and bin-length control the r-binning of g(r,t). Further, where this applies, the r-max value is reduced to be slightly less than half of the shorted box edge of atomic box specified with structure. Please note that calculate-rdf needs to be specified with additional elements when used with algorithms that compares with structural data only, as of this writing this means RDF (Radial Distribution Function). These are shown in the green comments and are cal-rdf-at-interval which sets the number of MD steps to move forward before calculating another RDF and average-over-this-many-rdf which sets the number of RDFs to average over to reduce statistical noise. Also, slightly confusing, apologies for this, when used with an algorithm that compares with RDF data, r-max and bin-length is specifying the binning of the RDF to save to file. Whereas the binning used for calculated RDF array that compare with the data is taken from the data (and r-max from the rdf-fom element discussed elsewhere).

When using the mdmc\_control\_time\_control algorithm the PE parameters are optimised against dynamical structure factor information, e.g. S(q, omega). It would make a lot of sense if the calculated S(q, omega) would automatically use the q and omega values specified in the data file. However, this has not been implemented yet, apologies, and currently these need to be specified explicitly using the notation:

<q-values>

<q start="0.42" step="0.42" n-step="2" />

<q start="1.62" step="0.3" n-step="2" />

<q start="2.58" step="0.42" n-step="2" />

<q start="3.9" step="10000.0" n-step="0" />

</q-values>

<omega-values>

<omega start="0.0" step="0.02" n-step="10" />

<omega start="0.25" step="0.05" n-step="15" />

<omega start="1.1" step="0.1" n-step="9" />

</omega-values>

## Other features of the job file

There is currently no user option for specifying an initial seed for the random number generator and therefore any run of MDMC will never match a previous run even when exactly the same job file is used.

## Partially implemented features

The MDMC code has been designed to accommodate constraints such as fixed neighbours constraints (FNC). Some skeleton code has been created for handling this but this feature is not fully implemented as of yet. The job file constraint notation which is not yet fully supported is:

<**constraints**>

<fnc-constraint filename="someFNCs.xml"></fnc-constraint>

<cutoff-constraint />

<coordination-constraint />

<soft-cutoff-constraint />

</constraints>

In a number of places in the job file the units attribute is used. As of this writing these must be specified as shown in the example in this user manual. It would be reasonably easy to extend this by adjusting the (XML) handler files in the MDMC code, see the technical documentation.

# The output

All algorithms output job\_log.txt, which contains logging information.

The algorithms mdmc\_control\_time\_corr and mdmc\_control output mdmc\_results.xml. This file contains an XML entry for each change to the PE parameters.

All remaining files outputted should be self-explanatory from the filename. For example, the output files first\_s\_q\_omega.xml and best\_s\_q\_omega.xml are the first and best S(q, omega) found. The code of each algorithm is fairly short and may (easily) be modified to return different outputs.

## Scripts for displaying the output

The folder for\_plotting\_results contains Matlab scripts for plotting and analysing MDMC outputs. These are: (warning: they have only been tested with argon data)

* analyseMDMCrun.m: reads mdmc\_results.xml and plots FOM as a function of MC steps for rejected and accepted moves.
* analyseMDMCcorrelation.m: reads mdmc\_results.xml and plots Lennard-Jones PE parameter values for the accepted moves to look for correlation etc.
* diff\_s\_q\_omega.m: plots the differences between two S(q, omega) files
* read\_einstein\_diffuse\_exp.m, read\_G\_d.m, read\_G\_s.m, read\_h\_d.m, read\_h\_s.m, read\_rdf.m, read\_s\_q\_omega.m and read\_s\_q\_time.m: read different MDMC data outputs and display such outputs

# References

Rapaport, D. C. (2004). *The art of molecular dynamics simulation*. Cambridge University Press.