MDQT README

Grant Gorman

October 3, 2019

1 Introduction

This document provides instructions for using the combined molecular dynamics (MD) and quantum trajectories (QT) code that simulates 1D optical molasses of the ions within a non-expanding, uniformly-distributed ultracold neutral plasma, as described in [1]. The MD portion of the code evolves each ion's position and velocity due to inter-ion forces derived from the Yukawa one-component plasma model. The QT portion of the code evolves the ion wavefunctions and x-velocities (along the cooling axis) according to the ion-light Hamiltonian, which includes the effects from a 408 nm cooling laser ($^2S_{1/2} \rightarrow {}^2P_{3/2}$) and a 1033 nm repump laser ($^2D_{5/2} \rightarrow {}^2P_{3/2}$).

These instructions assume the user has basic knowledge of the C++ language and the g++ compiler. Additionally, they assume the user is familiar with reference [1]. For simplicity, the MDQT code consists of a single file ('PlasmaMDQTSimulation.cpp'). Prior to running a simulation, the user must appropriately set the input parameters, which are contained in a clearly-labeled section in the first 100 lines of the MDQT code, and then the code must be compiled into an executable. Note that our simulation code is computationally expensive, so for best performance we suggest running it on a supercomputer and not a general-purpose computer.

The rest of this document is organized as follows. The MDQT input parameters and program options are discussed in Sec. 2. In Sec. 3 we provide instructions for compiling the code with the g++ compiler on a Linux-based computer. The output file architecture is described in Sec. 4, and Sec. 5 provides instructions for using our MATLAB analysis code that produces all plots found in [1]. Finally, Sec. 6 discusses how to continue a simulation from previously-saved conditions.

2 Input Parameters

This section provides a complete description of each simulation input parameter, including the use of each variable and its units. All input parameters are grouped into a clearly-labeled section within the first 100 lines of the MDQT code.

- saveDirectory (c-style string): Contains the directory in which simulation data will be saved. The directory is relative to the path of the executable file. Example: "dataFolder/".
- newRun (boolean): Tells the program whether to run a new simulation from random initial positions and zero velocity (true) or whether to continue a simulation from previously-saved conditions (false). See Sec. 6 for more details.

- c0Cont (integer): Only used when loading previously-saved conditions (e.g. newRun = false). c0Cont is a 6-digit integer that corresponds to the number of MD time steps undergone in the loaded simulation, and should match the 6-digit integer found in the previously-saved files (e.g. there is an output file with name 'ions_timestepXXXXXX.dat'). c0Cont should be set equal to XXXXXX. See Sec. 6 for more details.

- tmax (double): Time at which the simulation will end in units of $\omega_{pE}^{-1} = \sqrt{3}\,\omega_{pi}^{-1}$, where $\omega_{pi} = \sqrt{\frac{ne^2}{\epsilon_0 m}}$ is the plasma oscillation frequency. A new simulation will run from $t = 0 \rightarrow tmax$. A continued simulation, which starts at time t' (loaded from the save files), will run from $t = t' \rightarrow tmax$.

- density (double): Uniform, time-independent plasma density in units of $10^{14}\,\mathrm{m}^{-3}$.

- Ge (double): Electron Coulomb coupling parameter. We've chosen to define *density* and G_e as input parameters, so the electron temperature, T_e, is self-consistently defined later in the code according to $G_e = \frac{e^2}{4\pi\epsilon_0 a_{ws} k_B T_e}$, where $a_{ws} = (3/4\pi n)^{1/3}$ is the Wigner-Seitz radius. Typically, $Ge < .1$ to avoid three-body recombination.

- N0 (integer): Average number of particles within the simulation box. Note that the actual number of particles used within the simulation box is determined stochastically, and may differ slightly from N0. The actual particle number is contained within the variable N, which is saved at the end of the simulation (see Sec. 4).

- detuning (double): Detuning of the 408 nm cooling laser, in units of γ, that drives the $^2S_{1/2} \rightarrow {}^2P_{3/2}$ transition. $\gamma = 1.41 \times 10^8\,\mathrm{s}^{-1}$ is the natural linewidth of the cooling transition.

- detuningDP (double): Detuning of the 1033 nm repump laser, in units of γ, that drives the $^2D_{5/2} \rightarrow {}^2P_{3/2}$.

- Om (double): Rabi frequency of the 408 nm cooling laser in units γ.

- OmDP (double): Rabi frequency of the 1033 nm cooling laser in units γ.

- reNormalizewvFns (boolean): Program option that allows for forced renormalization of particle wavefunctions such that the total probability of each particle occupying a quantum state is one. This option is only used if necessary as a diagnostic tool while editing the QT code. This is set to false when running simulations with the working code.

- applyForce (boolean): Program option that allows the user to turn off the inter-ion interactions (false) or simulate ions that interact via the Yukawa potential (true). This can be used to simulate laser cooling of a collisionless plasma, as was done to obtain the $n = 0$ data in Fig. 6 of reference [1].

- vRange (double): Only used if applyForce = false. In this case, ion velocities are initialized between [-vRange,vRange] in units of m/s. In the absence of inter-ion interactions, it's useful to be able to initialize the plasma with a sufficiently wide thermal distribution. We're often interested in the ion state populations as a function of velocity. In the case that the Yukawa force is applied, the ions begin with zero velocity and subsequently gain kinetic energy as a result of disorder-induced heating. In this way, particles naturally span a velocity range that's useful for state population studies. With no inter-ion forces, however, the particle velocities remain relatively small because they only change due to the cooling/repump lasers. Thus, we must initialize the ions with velocities that span the range of interest.

- removeQuantumJump (boolean): Program option that allows the user to turn off the QT algorithm completely (true) or use it as normal (false). This is useful for simulating natural plasma evolution. This could also be done by setting Om = OmDP = 0, but turning it off altogether makes the code run faster.

- fracOfSig (double): Dimensionless parameter between 0 and 1 used for simulating laser cooling of plasmas in a moving frame, in some ways mimicking laser cooling of an expanding plasma. Although this does not account for the changes in n and T_e as a result of plasma expansion, it does provide information about how the cooling efficacy is affected by a plasma with a non-zero mean velocity, which Doppler-shifts ions with respect to the cooling/repump lasers. When $fracOfSig > 0$, we Doppler-shift the cooling/repump lasers by the expected expansion velocity for a plasma of size $sig0$ at a distance $fracOfSig * sig0$ from the plasma's center.

- sig0 (double): Initial RMS plasma radius in units of mm. This is only used if $fracOfSig \neq 0$. Note that $sig0$ does not represent the actual size of the plasma in the simulation, which is always uniformly-distributed. $sig0$ is only used to determine the hypothetical hydrodynamic expansion velocity of a plasma with initial size $sig0$.

- Time steps
  - QUANTUMTIMESTEP (double): Time step for QT algorithm in units ω_{pE}^{-1}.
  - DIHTIMESTEP (double): Time step for MD algorithm in units of ω_{pE}^{-1} that is used from $t = 0 \rightarrow tmaxDIH$ (see below).
  - TIMESTEP (double): Time step for MD algorithm in units of ω_{pE}^{-1} used from $t = tmaxDIH \rightarrow tmax$.
  - Note that the quantum timestep

- tmaxDIH (double): Time in units of ω_{pE}^{-1} at which we switch from using DIHTIMESTEP to TIMESTEP. If a single MD time step is desired, either set DIHTIMESTEP=TIMESTEP or set $tmaxDIH = 0$.

3 Compiling Instructions

The following instructions describe how to compile our MDQT program via the command line on a Linux computer using C++11. Prior to compiling, there are three modules that must be loaded:

1. Version 5.4.0 of the g++ compiler: This module may be loaded using the command 'module load GCC/5.4.0'.

2. Version 1.10.3 of OpenMPI: Our program uses parallelized loops to speed up expensive calculations (e.g. force calculations), and does so using Open MPI software. This module may be loaded using 'module load OpenMPI/1.10.3'.

3. Version 7.600.1 of the Armadillo library: This is used for linear algebra functionality and scientific computing. It may be loaded using 'module load Armadillo/7.600.1'.

Once all three modules have been loaded and the MDQT input parameters have been saved, the C++ code is now ready to be compiled into an executable using the g++ compiler. This may be done using the command 'g++ -std=c++11 -fopenmp -o runFile -03 MDQT.cpp -lm -larmadillo', which will compile the C++ program named 'MDQT.cpp' into an executable named 'runFile'. The

MDQT code may also be compiled using an integrated development environment so long as the correct versions of the above modules are loaded. We leave this up to the user.

Once the program has been compiled, the executable is ready to be submitted via the command line. The 'main' function within our C++ program accepts one integer input, which will denote a 'job' number. The job number is a unique integer that distinguishes one identical run of the program from another. Thus, the executable (with name 'runFile') may be submitted via the command line using 'runFile *int*', where *int* represents some integer read in by the program. This provides a straightforward way for submitting multiple instances of the code at once. Each time the program is run with a different job number, the output files will be saved within a folder named 'job*int*'. The next section will provide more details on how the output files are saved.

4 Output Files

The MDQT code saves important information about the plasma as a function of time. After a certain number of MD steps, the code records global information about the plasma, including the average ion kinetic energy, average potential energy per particle, and ion state populations. At the end of a simulation, particle positions, velocities, and wavefunctions are saved (these are loaded when continuing a simulation). For convenience, we have provided a MATLAB program that can generate the plots found in [1], which will be discussed in Sec. 5.

Before discussing the different save files, it's first important to understand how the folders are organized. The 'saveDirectory' input parameter provides a relative path to where data for a given simulation will be stored. If the executable is located within directory 'full/path/to/exec', then all data will be stored within 'full/path/to/exec/saveDirectory'. Within 'saveDirectory', a folder with name 'GeXXXDensityXXX...IonsXXXX' is saved. From now on, this will be referred to as the 'simulation data folder'. Each simulation data folder contains a job folder for each instance of the program that is run. Recall from Sec. 3 that the job number is the integer input parameter that the executable reads in to distinguish multiple runs of the same executable.

A description of each file saved by the MDQT program is discussed below. Each file type is saved within each job folder. We have provided a MATLAB program that processes and plots the data within these files (discussed in Sec. 5). Knowledge of these output files is not required to use the MATLAB program.

Each file below is saved within each job folder:

- energies.dat: Tab-delimited file whose columns contain energy-related information about the plasma as a function of time. Each recorded quantity is averaged over all particles. Time, energy, and velocity are recorded with units ω_{pE}^{-1}, $E_c = \frac{e^2}{4\pi\epsilon_0 a_{ws}}$, and $a_{ws}\omega_{pE}$, respectively. The columns are organized as $[t, KE_x, KE_y, KE_z, PE, PE(t) - PE(0), v_{exp,x}]$, where KE denotes kinetic energy, PE denotes potential energy, and $v_{exp,x}$ denotes mean x-velocity.
- statePopulationsVsVTimeXXXXXX.dat: Tab-delimited file containing the state populations for each ion as a function of the x-velocity (v_x). The columns are organized as follows: [v_x $P_s(v_x)$ $P_p(v_x)$ $P_d(v_x)$]. Each row corresponds to a different ion within the simulation. XXXXXX is a 6-digit integer that corresponds to the row number of the 'energies.dat' file, thus representing the time at which the state populations were recorded.
- vel_dist?_timeXXXXXX.dat: Contains the velocity distribution along a particular axis (? = x, y, or z) and particular time (XXXXXX corresponds to a row of 'energies.dat'). The first

column contains v_x and the second column contains the probability (relative to 1) of having that particular velocity.

- wvFns_timeXXXXXX.dat: This file contains the wavefunctions of all particles at the end of the simulation. This is read in when continuing a simulation. Each row corresponds to a different ion and each of the 24 columns correspond to twelve pairs of real/imaginary parts of the wavefunction coefficients. See Ref. [2] for wavefunction naming convention.

- conditions_timestepXXXXXX.dat: Tab-delimited file created at the end of the simulation that contains particle position and velocity information that is read in by the program when continuing a simulation. Each row corresponds to a different particle. The columns are structured as follows: $[x\ y\ z\ v_x\ v_y\ v_z]$. The positions and velocities are recorded with units of a_{ws} and $a_{ws}\,\omega_{pE}$, respectively. XXXXXX in the file name corresponds to the number of MD time steps undergone within the simulation up until this point.

- ions_timestepXXXXXX.dat: Tab-delimited file created at the end of the simulation that contains the particle number, the number of times the 'output' function was called (e.g. the number of MD time steps), and the simulation time the program ended at. It contains a single row with the aforementioned quantities contained in each column in the order of mention.

- simParams_timestepXXXXXX.dat: Tab-delimited file that contains all simulation input parameters used for this simulation. The first column of this file contains variable names and the second column contains the corresponding value used in the program with the same units.

5 Analyzing the Output Files

In addition to the MDQT simulation code, we have provided a MATLAB analysis code that plots the simulation data found within [1]. The user interface for this program is named 'mainSimAnalysis.m' and may be found within the 'Plasma-MDQT-Analysis' folder along with a folder of sub-programs called by the main function.

To run the program, open 'mainSimAnalysis.m' and ensure that the folder containing the sub-programs is added to the MATLAB search path, which may be done by modifying the 'addpath' command on line 7. Once that's done, press 'Run' and the program will allow the user to select one or more simulation folders via a dialogue box. MATLAB will then prompt the user to select which program options to use via the command window.

6 Continuing a Simulation

Due to the MDQT code being computationally expensive, you may run into a situation where the simulation will need to run longer than the time you're allotted in a single session. For example, some clusters may only allow you to run a simulation for 8 hours at a time, but in order to reach the desired $tmax$ it will take 10 hours. At the end of a simulation we record the ion positions, velocities, and wavefunctions. The code has the ability to continue a simulation by loading these previously-saved conditions.

It's important that the program finishes running without interruption because the last line of the code saves the particle conditions. If the program is terminated early, the particle conditions will not be saved and you will not be able to continue the simulation. How long the simulation takes depends

on the system it's run on, the number of particles, the density, and $tmax$. To obtain an estimate of how long the program will take to run, run a simulation with ∼3500 particles, $density = 2$, and $tmax < 1$, which should take less than an hour.

Assuming you have successfully completed a simulation, you may continue the simulation from the previously-saved conditions in the following way. First, except for 'newRun', 'c0Cont', and 'tmax', all the input parameters must be the same as they were for the original simulation. Then, set $newRun = false$ and set c0Cont equal to the 6-digit integer found within the most recent 'conditions_timestepXXXXXX.dat' file. Finally, remember that $tmax$ is not the duration of the simulation, it is the time at which the simulation ends. You must change $tmax$ to be greater than it was in the previous simulation, otherwise the continued simulation will end immediately.

Once the input parameters have been changed appropriately, save and recompile the C++ program following the instructions from Sec. 3. Make sure that the new executable file is contained within the same directory as the original executable file because the save directory is relative to the its location.

References

[1] G. M. Gorman, T. K. Langin, M. K. Warrens, and T. C. Killian. Combined molecular dynamics and quantum trajectories simulation of laser-driven, collisional systems. *Unknown*, ??:??, 2019.

[2] T. K. Langin. *Laser Cooling of Ions in a Neutral Plasma*. PhD thesis, 2019.