FFLUX testing protocol

Benjamin C. B. Symons

March 2022

1 Introduction

This document lays out a protocol that should be followed when any change is made to the DL_FFLUX code to ensure that no bugs have been introduced. The tests form a hierarchy that is useful when debugging i.e. by start at the bottom of the hierarchy and working up, one can hone in on the nature of a bug. However, the tests are structured such that, if the test at the top of the hierarchy is passed then it is safe to assume that everything is working.

When running the code in serial, testing is relatively simple as the code is deterministic and one can expect the numerical results (i.e. energies printed in the FFLUX file) to be identical before and after a change (assuming the code is compiled with same compiler, optimisation flags and is run on the same hardware). However, when parallelisation is enabled, the code is no longer deterministic and numerical differences are to be expected. In fact, when simulating on the bulk scale, small initial numerical differences will grow in magnitude and two different simulations with identical initial conditions will follow diverging trajectories through phase space. However, despite the trajectories diverging on the micro scale, the macro behaviour of the trajectories should be consistent. Even comparing macroscopic behaviour is difficult because this behaviour is characterised by a statistical ensemble i.e. you do not expect two trajectories to give identical macroscopic properties (e.g. density, diffusion coefficient) you would expect the ensemble average over a group of trajectories to be consistent with the ensemble average over a separate group of trajectories. Even this is not trivial because it is not necessarily clear how many trajectories of a given length are needed for a good average (it is highly system and property dependent). Proper averaging of this kind is likely to require 100's-1000's CPU hours and so it not feasible for unit testing. As such, some degree of judgement is required when carrying out these tests. This is the logic behind the bulk tests. Note that in all cases 'old' code will refer to the last known working version of the code i.e. the version without changes and 'new' will refer to the version with changes.

Note that when developing the code, it is also up to the developer to come up with suitable tests. For an example of this, see my paper on the development of smooth particle mesh Ewald. In the validation section of that paper, I outline how I used a combination of various mathematical limits of the smooth particle mesh Ewald sum alongside an alternate implementation of electrostatics in order to validate the code that I developed. These kinds of tests are highly useful as they validate certain algorithms within the code and so allow you to hone in on bugs/rule out sections of code when searching for a bug. Some further detail on these tests is given in the section on dimer tests.

2 Bulk tests

Each test should be done in serial and also in parallel to ensure that the changes have not broken the parallelisation. Note that there are also different levels of electrostatics L'=0,1,2,3,4. There are 3 sets of routines 1 handles L'=0, another L'=1,2 and the final set L'=3,4. If no changes have been made to the electrostatics code then tests can be done at any L'. If the electrostatics has been altered then test at the appropriate L'.

Two folders are provided on the GitHub in the unit tests directory: serial_water and parallel_water. The inputs are identical but the outputs will be slightly different as serial_water was run in serial and parallel_water was run on 8 cores using MPI. In terms of output files, the FFLUX file is provided which contains energies, the STATIS file is provided which provides a selection of properties (volume, pressure etc.) and the data for the RDF's are provided in the .dat files. These outputs can be used as things to compare against in tests 1-3.

2.1 Test 1: Energies

Firstly, using the inputs for test 1, run a single step calculation (i.e. set the number of steps to 0 in the CONTROL file) with the old and new codes. Compare the output of the FFLUX files. The energies printed there should all agree very well with the new and old codes. If running in serial then the agreement should be perfect or up to the last 1 or 2 decimal places. In parallel there might be a slightly larger discrepancy but the agreement should still be very good. It is very difficult to say exactly how good the agreement should be (hence the need for the later tests) but I would be suspicious of any discrepancy that occurs sooner than the last 3 or 4 decimal places. A failure in this test indicates a bug in the computation of energies. (see FFLUX file in examples).

2.2 Test 2: Forces

Repeat the previous test but now set the number of steps in CONTROL to 1 (i.e. a total of two steps). The second row of energies in the FFLUX file should agree well (although less well than the first line if in parallel). For the energies to be correct in the second step of the simulation, both the energy and force calculations must be correct. (see FFLUX file in examples).

2.3 Test 3: Full simulation

Even if tests 1A and 1B are passed, there is still the possibility of a bug. It could be that there is a bug that will only manifest later down the line, or one that grows in magnitude over time meaning it is not apparent from a very short simulation. Given the difficulty of comparing raw numerical results between bulk simulations as discussed in the introduction, we must compare a macroscopic property computed from the two trajectories in order to be certain they are behaving consistently.

The property we will compute is the radial distribution function (RDF) which relates to the structure of the liquid. For water there are 3 RDF's: OO, OH and HH (where O and H are oxygen and hydrogen respectively). This property is a nice test because it converges very quickly. The RDF (.dat) files provided in serial_water and parallel_water should be

used as the 'truth' in the test. Both the serial and parallel RDFs are provided in order to give an idea about the degree of discrepancy that is to be reasonably expected when changing the code/running on a different system i.e. if the discrepancy between the RDFs produced by a changed code is considerably larger than the discrepancy between the two sets of provided RDFs then it is a likely indication of a bug.

The RDF's are computed using the software VMD. In order to compute it, the HISTORY file must be loaded into VMD (select the option "DL_POLY_4 HISTORY". In order to compute RDF's using VMD, the radial distribution function window can be brought up by selecting "Extensions" followed by "Analysis" and finally "Radial Pair Distribution Function g(r)". Once the RDF window has been selected, use the input parameters shown in Figure 1.

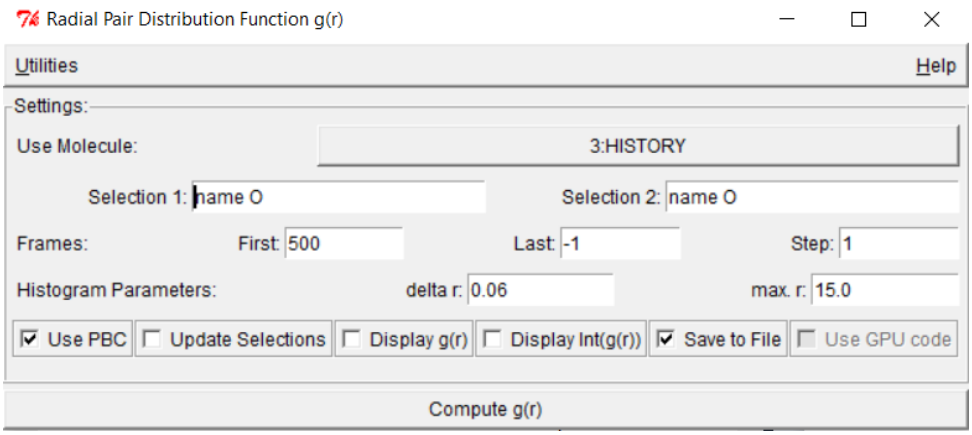

Figure 1: Screenshot of the VMD radial distribution function menu.

In order to compute the OH RDF, one of the selection fields in Figure 1 should be changed to "name H". To compute the HH RDF, both selection fields should be changed to "name H". As with all of the tests so far, a certain degree of judgement is required but, the RDFs should agree essentially perfectly visually. An example is shown in Figure 2.

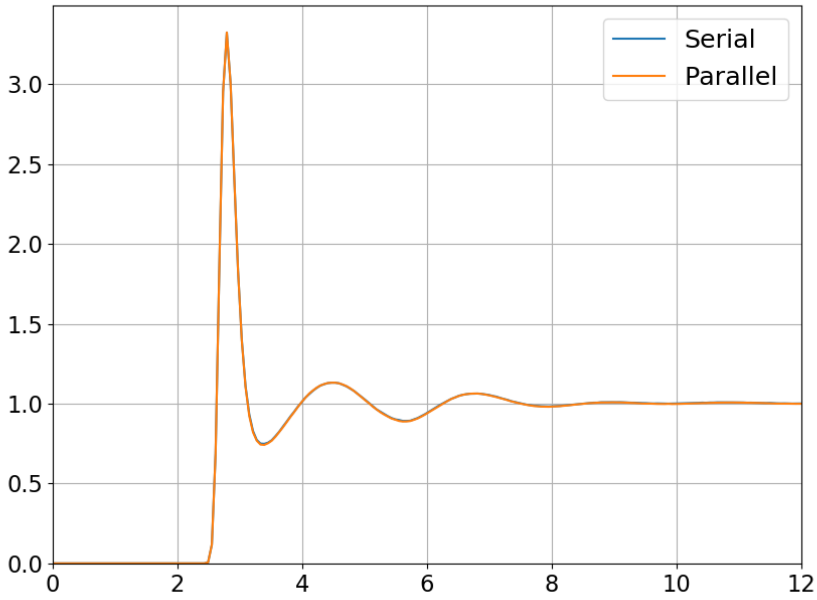

Figure 2: Comparison of oxygen-oxygen RDFs computed from a serial and parallel run.

A successful test should look like Figure 2 i.e. almost no discernible difference assuming the RDFs have converged properly. In Figure 2, the runs even had different initial conditions as well as being serial and parallel. Figure 3 shows the absolute difference between the two RDF's at each point. Sample RDF data files (.dat files) are provided on the GitHub which give an idea of the expected deviation.

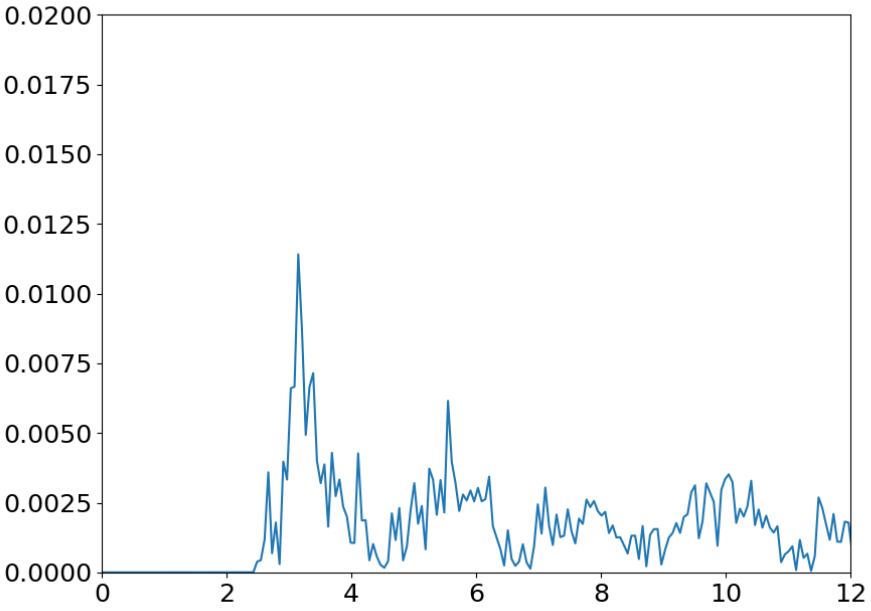

Figure 3: Absolute difference between the RDFs shown in Figure 2.

If the RDF's agree then it is highly likely the changes to the code have not introduced a bug. However, I have seen once before that there was a bug that kept the structure of the water correct i.e. the RDFs looked good but, the water had an overall translation. In this case, there was a discrepancy in the RDF's that was visible in a plot like Figure 2 but it was still relatively small. As such, it is necessary to be careful when evaluating whether or not a test has passed/failed.

2.4 Test 4: formamide

Water is actually a special case because it is a system with 3 features. In this special case, there are parts of the code that are never used. As such it is necessary to also run tests on a system that is larger than water. Ideally, tests would always be done with a larger system that uses all the code but, at the time of writing, no bulk properties have ever been computed with DL_FFLUX for any system other than water. As such, tests 1 and 2 can be replicated for formamide (chosen simply because it is a system that we have good ML models for). Test 3 cannot yet be replicated. Instead, a less rigorous test can be done where a longer simulation (10,000+ steps) is run and the trajectory checked to make sure nothing untoward happens i.e. no explosions etc. A set of example formamide input files is provided in the formamide_test directory.

3 Dimer tests

These tests are discussed in my smooth particle mesh Ewald (SPME) paper. However, some further details are given here. In principle the electrostatics code should not need further development and so these tests should just serve as an example rather than unit tests that need to be conducted on a regular basis. These tests are only for advanced developers as they require changes to be made to the code to carry out.

The tests exploit the fact that the Ewald sum in a certain limit that is referred to as the real space limit (see paper for more details) is identical to the Coulomb sum. Both SPME and the Coulomb sum are implemented in DL_FFLUX (Ewald and cluster methods respectively). This means that within DL_FFLUX there is the capacity to use two different methods for evaluating an electrostatic interaction that should give out identical answers. This is especially use as the electrostatics can be broken down on a term by term basis i.e. the monopole-monopole can be separated from the monopole-dipole etc. This is ideal from a bug hunting perspective. However, the only difficulty is that the two methods: SPME and Coulomb sum, differ in their choice of which interactions they compute at a given level L'/L = n. SPME utilises the so-called L' formalism where L'= n activates all multipole moments up to and including rank n and then computes all possible interactions with those multipole moments i.e. for L'=1, charge-charge, charge-dipole and dipole-dipole are computed. The Coulomb sum code on the other hand uses the L formalism which behaves differently. For a given L = n, all interactions between all moments on atom A and atom B that satisfy the relation $L = l_A + l_B + 1$ are evaluated. This is terminates the expansion based on inverse powers of R rather than on which moments are enabled. The L formalism computes fewer interactions at a given level than the L' formalism. It is possible to tweak the Coulomb sum code (fflux_calc_mp_forces.f90) in order to choose exactly which interactions are computed i.e. to manually get around the

L formalism and match it to the L'. This kind of manual tweaking is necessary to perform the dimer tests outlined here.