1 Introduction

1.1 Overview

This introduction provides a brief overview of what the *Ludwig* code does, how to obtain and build it, and how to run a sample problem. It is assumed that the reader has at least a general knowledge of Navier-Stokes hydrodynamics, complex fluids, and to some extent statistical physics. This knowledge will be required to make sense of the input and output involved in using the code. Those wanting to work on or develop the code itself will need knowledge of ANSI C, and perhaps message passing and CUDA. We assume the reader is using a Unix-based system.

1.1.1 Purpose of the code

The *Ludwig* code has been developed over a number of years to address specific problems in complex fluids. The underlying hydrodynamic model is based on the lattice Boltzmann equation (LBE, or just 'LB'). This itself may be used to study simple (Newtonian) fluids in a number of different scenarios, including porous media and particle suspensions. However, the code is more generally suited to complex fluids, where a number of options are available, among others: symmetric binary fluids and Brazovskii smectics, polar gels, liquid crystals, or charged fluid via a Poisson-Boltzmann equation approach. These features are added in the framework of a free energy approach, where specific compositional or orientational order parameters are evolved according to the appropriate coarse-grained dynamics, but also interact with the fluid in a fully coupled fashion.

A number of other features are catered for is some or all situations. These include fluctuating hydrodynamics, and appropriate fluctuating versions of order parameter dynamics for binary solvents and liquid crystals. Colloidal particles are available for simulation of suspensions and dispersions, and with appropriate boundary conditions binary mixtures, liquid crystals, and charged fluid. Shear flow is available for fluid-only systems implemented via Lees-Edwards sliding periodic boundaries. Not all these features play with all the others: check the specific section of the document for details.

Broadly, the code is intended for complex fluid problems at low Reynolds numbers, so there is no consideration of turbulence, high Mach number flows, high density ratio flows, and so on.

1.1.2 How the code works

We aim to provide a robust and portable code, written in ANSI C, which can be used to perform serial and scalable parallel simulations of complex fluid systems based around hydrodynamics via the lattice Boltzmann method. Time evolution of modelled quantities takes place on a fixed regular discrete lattice. The preferred method of dealing with the corresponding order parameter equations is by using finite difference. However, for the case of a binary fluid, a two-distribution lattice Boltzmann approach is also maintained for historical reference.

Users control the operation of the code via a plain text input file; output for various data are available. These data may be visualised using appropriate third-party software (e.g., Paraview). Specific diagnostic output may require alterations to the code.

Potential users should also note that the complex fluid simulations enabled by *Ludwig* can be time consuming, prone to instability, and provide results which are difficult to interpret. We make no apology for this: that's the way it is.

1.1.3 Who should read what?

This documentation is aimed largely at users, but includes information that will also be relevant for developers. The documentation discusses the underlying features of the coordinate system, hydrodynamics (LB), and other generic features, and has a separate section for each free energy. You will need to consult the section relevant to your problem.

1.2 Obtaining the code

Under the auspices of the CCPs (Collaborative Computational Projects), and in particular CCP5, we have a CCPForge project account at `http://ccpforge.cse.rl.ac.uk/` which provides revision control via SVN, issue tracking, and so on.

Users: Code releases may be downloaded via anonymous download

```
http://ccpforge.cse.rl.ac.uk/gf/project/ludwig/frs/
```

The current release is available as `ludwig-0.4.0.tar.gz`. Alternatively, registered users can obtain the code via svn checkout in the same way as developers.

Developers: Developers need to create an account at CCPForge, and let one of the project owners[1] know the user name so you can be added to the access list.

To download the entire code, developers need

```
svn checkout --username <user> http://ccpforge.cse.rl.ac.uk/svn/ludwig
```

where `<user>` is replaced by your CCPForge user name. You will see

```
$ ls ludwig
branches tags     trunk
```

which is the usual svn convention for trunk, branches and tags. The process for committing code is described in section 3.5.

If you look in the `trunk` directory, you should see

```
$ ls trunk
config LICENSE     mpi_s  src      tests  version.h
docs   Makefile.mk README targetDP util
```

[1]Currently: `kevin@epcc.ed.ac.uk`, `ohenrich@epcc.ed.ac.uk`

2 Quick Start for Users

This section provides a brief overview of compiling and running the code, which should allow users to get off the ground. The section assumes the code has been downloaded and you are in the top level directory.

2.1 Configuration

You will need to copy one of the existing configuration files from the `config` directory to the top level directory. This file sets all the relevant local details for compilation and so on. For example:

```
$ cp config/lunix-gcc-default.mk config.mk
```

Edit the new `config.mk` file to check, or adjust, the options as required. You will need to identify your local C compiler. The example uses `gcc` in serial, and `mpicc` in parallel. For MPI in particular, local details may vary. A quick way to compile the code is via

```
$ cd tests
$ make compile-mpi-d3q19
```

This integrates a number of separate stages which are described indivually in the following sections. The resulting executable will be `./src/Ludwig.exe` at the top level.

2.1.1 targetDP

A `targetDP` layer is required by the code in all cases. It allows compilation for both simple CPU systems and for a number of accelerator systems using a single source code. This will make the appropriate choices depending what has been specified in the top-level `config.mk` file. From the top-level directory

```
$ cd targetDP
$ make
```

should produce the appropriate `libtarget.a` library.

2.1.2 Serial

First, compile the MPI stub library in the `mpi_s` directory. You should be able to build the library and run the tests using, e.g.:

```
$ make libc
gcc -Wall -I. -c mpi_serial.c
ar -cru libmpi.a mpi_serial.o
$ make testc
gcc -Wall  mpi_tests.c -L. -lmpi
./a.out
Running mpi_s tests...
Finished mpi_s tests ok.
```

Now compile the main code in the `src` directory. Compilation should look like:

```
$ make serial
make serial-d3q19
make serial-model "LB=-D_D3Q19_" "LBOBJ=d3q19.o"
make lib
make libar "INC=-I. -I../mpi_s" "LIBS= -L../mpi_s -lmpi -lm"
gcc -D_D3Q19_ -Wall -O2 -I. -I../mpi_s -c model.c
gcc -D_D3Q19_ -Wall -O2 -I. -I../mpi_s -c propagation.c
...
```

which shows that the default lattice Boltzmann model is D3Q19. Successful compilation will provide an executable `Ludwig.exe` which is linked against the MPI stub library.

2.1.3 Parallel

Here, you do not need to compile the stub library. Simply compile the main code with, e.g.,:

```
$ make mpi
make mpi-d3q19
make mpi-model "LB=-D_D3Q19_" "LBOBJ=d3q19.o"
make libmpi
make libar "CC=mpicc"
mpicc -D_D3Q19_ -Wall -O2 -I. -c model.c
mpicc -D_D3Q19_ -Wall -O2 -I. -c propagation.c
...
```

Again, this should produce an executable `./Ludwig.exe` in the current directory, this time linked against the true MPI library.

2.2 Running an Example

2.2.1 Input file

The behaviour of *Ludwig* at run time is controlled by a plain text input file which consists of a series for key value pairs. Each key value pair controls a parameter or parameters within the code, for example the system size and number of time steps, the fluid properties density and viscosity, the free energy type and associated parameters (if required), frequency and type of output, parallel decomposition, and so on.

For most keys, the associated property has some default value which will be used by the code if the relevant key is not present (or commented out) in the input file. While such default values are chosen to be at least sensible, users need to be aware that all necessary keys need to be considered for a given problem. However, many keys are irrelevant for any given problem.

A significant number of example input files are available as part of the test suite, and these can form a useful starting point for a given type of problem. We will consider one which uses some of the common keys.

2.2.2 Example input

We will consider a simple example which computes the motion of a single spherical colloidal particle in an initially stationary fluid of given viscosity. The velocity may be measured as a function of time. Assume we have an executable in the `src` directory.

Make a copy of the input file

```
$ cp ../tests/regression/d3q19/serial-auto-c01.inp .
```

Inspection of this file will reveal the following: blank lines and comments — lines beginning with `#` — may be included, but are ignored at run time; other lines are parsed as key value pairs, each pair on a separate line, and with key and value separated by one or more spaces. Values may be characters or strings, a scalar integer or 3-vector of integers, or a scalar or 3-vector of floating point numbers. Values which are 3-vectors are delineated by an underscore character (not a space), e.g., `1_2_3`, and always correspond to a Cartesian triple (x, y, z). A given value is parsed appropriately in the context of the associated key.

So, the first few lines of the above file are:

```
##############################################################################
#
#  Colloid velocity autocorrelation test (no noise).
#
##############################################################################

N_cycles 40

##############################################################################
#
#  System and MPI
#
##############################################################################

size 64_64_64
grid 2_2_2
```

Here, the first key value pair `N_cycles 40` sets the number of time steps to 40, while the second, `size 64_64_64`, sets the system size to be 64 points in each of the three coordinate directions. The `grid` key relates to the parallel domain decomposition as is discussed in the following section.

2.2.3 Run time

The executable takes a single argument on the command line which is the name of the input file, which should be in the same directory as the executable. If no input file name is supplied, the executable will look for a default `input`. If no input file is located at all, an error to that effect will be reported.

Serial: If a serial executable is run in the normal way with the copy of the input file as the argument the code should take around 10–20 seconds to execute 40 steps. The fist few lines of output should look like:

```
$ ./Ludwig.exe ./serial-auto-c01.inp
Welcome to Ludwig v0.2.16 (Serial version running on 1 process)

The SVN revision details are: 2638M
Note assertions via standard C assert() are on.

Read 20 user parameters from serial-auto-c01.inp

No free energy selected

System details
--------------
System size:  64 64 64
```

```
Decomposition: 1 1 1
Local domain:  64 64 64
Periodic:      1 1 1
Halo nhalo:    1
```

This output shows that the appropriate input file has been read, and the system size set correspondingly with a number of default settings including periodic boundary conditions. "No free energy" tells us we are using a single Newtonian fluid.

Normal termination of execution is accompanied by a report of the time take by various parts of the code, and finally by

```
...
Ludwig finished normally.
```

Parallel: The executable compiled and linked against the MPI library can be run with the MPI launcher for the local system, often `mpirun`. For example, a run on 8 MPI tasks produces a similar output to that seen in the serial case, but reports details of the local domain decomposition:

```
$ mpirun -np 8 ./Ludwig.exe ./serial-auto-c01.inp
Welcome to Ludwig v0.1.26 (MPI version running on 8 processes)
...
System details
--------------
System size:   64 64 64
Decomposition: 2 2 2
Local domain:  32 32 32
...
```

The decomposition is controlled by the `grid` key in the input. Here, `grid 2_2_2` is consistent with the 8 MPI tasks specified on the command line, and the resulting local decomposition is 32 lattice points in each coordinate direction. If no grid is specified in the input, or a request is make for a grid which cannot be met (e.g., the product of the grid dimensions does not agree with the total number of MPI tasks available) the code will try to determine its own decomposition as best it can. If no valid parallel decomposition is available at all, the code will exit with a message to that effect.

Further details of various input key value pairs are given in relevant sections of the documentation.

2.2.4 Output

The standard output of the running code produces a number of aggregate quantities which allow a broad overview of the progress of the computation to be seen by the user. These include, where relevant, global statistics related to fluid density and momentum, the integrated free energy, particle-related quantities, and so on. The frequency of this information can be adjusted from the input file (see, e.g., `freq_statistics`).

Output for lattice-based quantities (for example, the velocity field $u(\mathbf{r};t)$) is via external file. This output is in serial or in parallel according to how the model is run, and may be in either ASCII or raw binary format as required. Output files are produced with time step encoded in the file, and an extension which describes the parallel output decomposition. Further, each quantity output in this way is accompanied by a metadata description with `meta` extension. For example, the two output files

```
bash-3.2$ ls dist*
dist-00000020.001-001 dist.001-001.meta
```

contain the LB distributions for time step 20 (the file is 1 of a total number of 1 in parallel), and the plain text metadata description, respectively.

To ensure output for based-based quantities is in the correct order for analysis, post-processing may be required (always if the code is run in parallel). This uses a utility provided for the purpose which required both the data and the metadata description to recombine the parallel output. This utility is described ELSEWHERE.

Output for colloidal particle data is again to file with a name encoding the time step and parallel docomposition of the output. For example,

```
bash-3.2$ ls config*
config.cds00000020.001-001
```

contains particle data for time step 20 in a format described ELSEWHERE. These data may be requested in ASCII or raw binary format.

2.2.5 Errors and run time failures

The code attempts to provide meaningful diagnostic error messages for common problems. Such errors include missing or incorrectly formatted input files, inconsistent input values, and unacceptable feature combinations. The code should also detect other run time problems such as insufficient memory and errors writing output files. These errors will result in termination.

Instability in the computation will often be manifested by numerically absurd values in the statistical output (ultimately `NaN` in many cases). Instability may or may not result in termination, depending on the problem. Such instability is very often related to poor parameter choices, of which there can be many combinations. Check the relevant section of the documentation for advice on reasonable starting parameters for different problems.

2.3 Note on Units

All computation in *Ludwig* is undertaken in "lattice units," fundamentally related to the underlying LB fluid model which expects discrete model space and time steps $\Delta x = \Delta t = 1$. The natural way to approach problems is then to ensure that appropriate dimensionless quantities are reasonable. However, "reasonable" may not equate to "matching experimental values". For example, typical flows in colloidal suspensions my exhibit Reynolds numbers as low as $O(10^{-6})$ to $O(10^{-8})$. Matching such a value in a computation may imply an impractically small time step; a solution is to let the Reynolds number rise artificially with the constraint that it remains small compared to $O(1)$. Further discussion of the issue of units is provided in, e.g., [7]. Again, consult the relevant section of the documentation for comments on specific problems.

3 General Information for Users and Developers

This section contains information on the design of the code, along with details of compilation an testing procedures which may be of interest to both users and developers. *Ludwig* is named for Ludwig Boltzmann (1844–1906) to reflect its background in lattice Boltzmann hydrodynamics.

3.1 Manifest

The top level ludwig directory should contain at least the following:

```
$ ls
bash-3.2$ ls
config LICENSE     mpi_s  src       tests  version.h
docs   Makefile.mk README targetDP util
```

The code is released under a standard BSD license; the current version number is found in `version.h`. The main source is found in the `src` directory, with other relevant code in `tests` and `util`. The `targetDP` directory holds code related to the targetDP abstraction layer [17]. These documents are found in `docs`.

3.2 Code Overview

The code is ANSI C (1999), and can be built without any external dependencies, with the exception of the message passing interface (MPI), which is used to provide domain-decomposition based parallelism. Note that the code will also compile under NVIDIA `nvcc`, i.e., it also meets the C++ standard.

3.2.1 Design

The *Ludwig* code has evolved and expanded over a number of years to its present state. Its original purpose was to investigate specifically spinodal decomposition in binary fluids (see, e.g., [23]). This work used a free-energy based formulation combined with a two-distribution approach to the binary fluid problem (with lattice Boltzmann model D3Q15 at that time). This approach is retained today, albeit in a somewhat updated form. The wetting problem for binary fluid at solid surfaces has also been of consistent interest. The code has always been developed with parallel computing in mind, and has been run on a large number of different parallel machines including the Cray T3E at Edinburgh. These features, developed by J.-C. Desplat and others, were reflected in the early descriptive publication in *Comp. Phys. Comm* in 2001 [11].

The expansion of the code to include a number of additional features has occasioned significant alterations over time, and little of the original code remains. However, the fundamental idea that the code should essentially operate for high performance computers and be implemented using ANSI C with message passing via MPI has remained unchanged.

The code has a number of basic building blocks which are encapsulated in individual files.

3.2.2 Parallel environment

The code is designed around message passing using MPI. A stub MPI library is provided for platforms where a real MPI is not available (an increasingly rare occurrence), in which case the code runs in serial. The parallel environment (interface defined in `pe.h`) therefore underpins the entire code and provides basic MPI infrastructure such as `info()`, which provides `printf()` functionality at the root process only. The parallel environment also provides information on version number etc. MPI parallelism is implemented via standard domain decomposition based on the computational lattice (discussed further in the following section on the coordinate system).

3.2.3 targetDP: threaded parallelism and vector parallelism

To allow various threaded models to be included without duplicating source code, we have developed a lightweight abstraction of the thread level parallelism, This currently supports a standard single-threaded model, OpenMP, or CUDA. targetDP ("target data parallel") allows single kernels to be written which can then be compiled for the different threaded models which may be appropriate on different platforms. targetDP can also be used as a convenient way to express vector parallelism (typically SSE or AVX depending on processor architecture). targetDP allows explicit specification of the vector length at compile time.

targetDP does not automatically identify parallelism; this must still be added appropriately by the developer. It is merely a concise way to include different threaded models. It is only available in the development version.

3.2.4 Coordinate system

It is important to understand the coordinate system used in the computation. This is fundamentally a regular, 3-dimensional Cartesian coordinate system . (Even if a D2Q9 LB model is employed, the code is still fundamentally 3-dimensional, so it is perhaps not as efficient for 2-dimensional problems as it might be.)

The coordinate system is centred around the (LB) lattice, with lattice spacings $\Delta x = \Delta y = \Delta z = 1$. We will refer, in general, to the lattice spacing as Δx throughout, its generalisation to three dimensions x, y, z being understood. Lattice sites in the $x-$direction therefore have unit spacing and are located at $x = 1, 2, \ldots, N_x$. The length of the system $L_x = N_x$, with the limits of the computational domain begin $x = 1/2$ and $x = L_x + 1/2$. This allows us to specify a unit control volume centred on each lattice site x_i being $i - 1/2$ to $i + 1/2$. This will become particularly significant for finite difference (finite volume) approaches discussed later.

Information on the coordinate system, system size and so on is encapsulated in `coords.c`, which also deals with the regular domain decomposition in parallel. Decompositions may be explicitly requested by the user in the input. or computed by the code itself at run time. `coords.h` also provides basic functionality for message passing within a standard MPI Cartesian communicator, specification of periodic boundary conditions, and so on.

Three-dimensional fields are typically stored on the lattice, but are addressed in compressed one-dimensional format. This avoids use of multidimensional arrays in C. This addressing must take account of the width of the halo region required at the edge of each sub-domain required for exchanging information in the domain decomposition. The extent of the halo region varies depending on the application required, and is selected automatically at run time.

3.2.5 Lattice Boltzmann hydrodynamics

The hydrodynamic core of the calculation is supplied by the lattice Boltzmann method, which was central at the time of first development. LB is also the basis for hydrodynamic solid-fluid interactions at stationary walls and for moving spherical colloids. The LB approach is described in more detail in Section 5. For general and historical references, the interested reader should consider, e.g., Succi [37].

3.2.6 Free energy

For complex fluids, hydrodynamics is augmented by the addition of a free energy for the problem at hand, expressed in terms of an appropriate order parameter. The order parameter may be a three dimensional field, vector field, or tensor field. For each problem type, appropriate tome evolution of the order parameter is supplied.

Coupling between the thermodynamic sector and the hydrodynamics is abstracted so that the hydrodynamic core does not require alteration for the different free energies. This is implemented via a series of call back functions in the free energy which are set appropriately at run time to correspond to that specified by the user in the input.

Different (bulk) fluid free energy choices are complemented by appropriate surface free energy contributions which typically alter the computation of order parameter gradients at solid surface. Specific gradient routines for the calculation of order parameter gradients may be selected or added by the user or developer.

Currently available free energies are:

- Symmetric binary fluid with scalar compositional order parameter $\phi(\mathbf{r})$ and related Cahn-Hilliard equation;
- Brazovskii smectics, again which scalar compositional order parameter $\phi(\mathbf{r})$;
- Polar (active) gels with vector order parameter $P_\alpha(\mathbf{r})$ and related Leslie-Erikson equation;
- Landau-de Gennes liquid crystal free energy with tensor orientational order parameter $Q_{\alpha\beta}(\mathbf{r})$, extended to apolar active fluids and related Beris-Edwards equation;
- a free energy appropriate for electrokinetics for charged fluids and related Nernst-Planck equations (also requiring the solution of the Poisson equation for the potential);
- a coupled electrokinetic binary fluid model;
- a liquid crystal emulsion free energy which couples a binary composition to the liquid crystal.

See the relevant sections on each free energy for further details.

3.3 Compilation

Compilation of the main code is controlled by the `config.mk` in the top-level directory. A number of example `config.mk` files are provided in the `config` directory (which can be copied and adjusted as needed). This section discusses a number of issues which influence compilation.

3.3.1 Dependencies on third-party software:

There is the option to use PETSC to solve the Poisson equation required in the electrokinetic problem. A rather less efficient in-built method can be used if PETSC is not available. We suggest using PETSC v3.4 or later available from Argonne National Laboratory `http://www.mcs.anl.gov/petsc/`.

The tests use the lightweight implementation of exceptions provided under the GPL Lesser General Public License by Guillermo Calvo. This is included with the source.

3.3.2 Makefile

The `Makefile.mk` and `config.mk` files in the top level directory control options for compilation. Before compilation, the `config.mk` file must be edited provide details of the local C compiler(s). The relevant lines are usually limited to, e.g.:

```
CC = cc
MPICC = mpicc
CFLAGS = -O2
```

where `CC` is the compiler used for serial compilation, `MPICC` is the compiler used for compilation against MPI, and `CFLAGS` provides compiler switches (often related to optimisation).

The `Makefile` provided in each subdirectory includes the `config.mk` file via the top level `Makefile.mk`. Changes to the individual Makefiles should therefore not be required.

If GPU compilation is required, the `config.mk` file should specify the local details concerning `nvcc`. This includes explicit information on the location of MPI headers and libraries if an MPI-parallel GPU version is required.

3.3.3 C assertions

The code makes quite a lot of use of standard C assertions, which are useful to prevent errors. They do result in a considerably slower execution in some instances, so production runs should switch off the assertions with the standard `NDEBUG` preprocessor flag. Add `-DNDEBUG` to `CFLAGS` in the `config.mk`.

3.3.4 Targets for serial and parallel code

The code can be compiled with or without a true MPI library. For serial execution, the MPI stub library must be compiled first (see “Quick Start for Users” Section 2.1.2). The appropriate target is:

```
bash-3.2$ make serial
make serial-d3q19
make serial-model ‘‘LB=-D_D3Q19_’’ ‘‘LBOBJ=d3q19.o’’
...
```

from which it will be seen that the default LB model is D3Q19. This is the same for the true MPI target

```
bash-3.2$ make mpi
make mpi-d3q19
make mpi-model ‘‘LB=-D_D3Q19_’’ ‘‘LBOBJ=d3q19.o’’
...
```

3.3.5 Targets for different LB models

Specific targets are provided if an alternative LB model is wanted. The available options are:

```
make [ serial-d2q9 | serial-d3q15 | serial-d3q19 ]
make [ mpi-d2q9 | mpi-d3q15 | mpi-d3q19 ]
```

A further set of targets is supplied if ‘reverse’ or structure-of-array memory ordering is wanted (e.g., for GPU computing):

```
make [ serial-d2q9r | serial-d3q15r | serial-d3q19r ]
make [ mpi-d2q9r | mpi-d3q15r | mpi-d3q19r ]
```

This will only influence efficiency of the code: results are unchanged.

3.4 Tests

A series of tests are provided in the `./tests` directory. These are of two types. Unit tests are generally written when units of code are first introduced to ensure basic operation is error free. The unit tests are encoded in an executable linked against the appropriate *Ludwig* library. Regression tests are introduced and updated when physical results are (broadly) validated. The regression tests use the stand-alone executable for the appropriate model, and read input files which define the different tests. Both are run regularly as the basis of a nightly test procedure, but they can be run independently, e.g., after adding or changing code.

3.4.1 Running the tests

A number of different `Makefile` targets are provided for running serial, parallel, or GPU tests. For each test, there is the option to run either the relevant unit tests, or regression tests, or both. For example, to run both serial unit tests and serial regression tests for the D3Q19 model, invoke

```
$ cd tests
$ make compile-run-serial
```

in the test directory. The model selection follows the naming scheme described for compilation in the previous section.

3.5 Additional Notes for Developers

Developers who wish to contribute code to the SVN repository please consider the following pleas concerning standards.

3.5.1 Coding standards

While definitive statements on style are avoided, please try to maintain some basic standards: (0) code should be strictly ANSI C99 standard; (1) code should compile without warnings when appropriate compiler flags are set (e.g., `-Wall` under `gcc`); (2) avoid long lines of code for readability reasons; (3) avoid confusing commented-out code; (4) avoid conditional pre-processor directives if possible; (5) add meaningful and descriptive comments; (6) use standard `assert()` to trap programming errors; (7) explicitly trap possible run time errors (8) add appropriate tests.

3.5.2 Documentation standards

Additions and alterations to the code need to be reflected in the documentation. These should be checked in at the same time as the code itself.

3.5.3 Protocol

Before checking in code, please follow procedure: (1) increment the patch version number in `version.h` consistent with the SVN (if a minor version increment is required, communication with other developers must be considered); (2) run at least the unit tests and the short regression tests; (3) add a note to the change log; and (4) SVN update and commit.

4 Further Information for Developers

4.1 Address models for 3-dimensional fields

The code allows for different order of storage associated with 3-dimensional fields (scalars, vectors, and so on). For historical reasons these are referred to as ‘Model’ and ‘Model R’ options, which correspond to array-of-structures and structure-of-arrays layout, respectively. Early versions of the code for CPU were written to favour summation of LB distributions on a per lattice site basis in operations such as $\rho(\mathbf{r}) = \sum_i f_i(\mathbf{r})$. This is array-of-structures, where the f_i are stored contiguously per site. Introduction of GPU versions, where memory coalescing favours the opposite memory order, were then referred to as ‘Model R’, the ‘R’ standing for ‘reverse’.

The memory layouts are discussed below. In all cases, a 3-d field occupies lattice sites with a unique spatial index determine by position, and computed via `coords_index()`. These position indices will be denoted $r_0, r_1, \dots, r_{n_s-1}$ where n_s is the total number of lattice sites (including halo points).

4.1.1 Rank 1 objects (to include scalar fields)

ADDR_MODEL: The array-of-structures or Model order for an $n-$vector field with components $v_\alpha = (v_0, v_1, \dots, v_{n-1})$ is, schematically:

$$\underbrace{\begin{array}{|c|c|c|c|}\hline v_0 & v_1 & \dots & v_{n-1}\\\hline\end{array}}_{r_0}\underbrace{\begin{array}{|c|c|c|c|}\hline v_0 & v_1 & \dots & v_{n-1}\\\hline\end{array}}_{r_1}\underbrace{\begin{array}{|c|c|c|c|}\hline v_0 & v_1 & \dots & v_{n-1}\\\hline\end{array}}_{r_2}\cdots\underbrace{\begin{array}{|c|c|c|c|}\hline v_0 & v_1 & \dots & v_{n-1}\\\hline\end{array}}_{r_{n_s-1}}$$

ADDR_MODEL_R: The Model R version is:

$$\overbrace{\begin{array}{cccc}\begin{array}{|c|c|c|c|}\hline v_0 & v_0 & \dots & v_0\\\hline\end{array}\\ r_0 \quad r_1 \qquad r_{n_s-1}\end{array}}^{component_0}\overbrace{\begin{array}{cccc}\begin{array}{|c|c|c|c|}\hline v_1 & v_1 & \dots & v_1\\\hline\end{array}\\ r_0 \quad r_1 \qquad r_{n_s-1}\end{array}}^{component_1}\cdots\overbrace{\begin{array}{cccc}\begin{array}{|c|c|c|c|}\hline v_{n-1} & v_{n-1} & \dots & v_{n-1}\\\hline\end{array}\\ r_0 \quad r_1 \qquad r_{n_s-1}\end{array}}^{component_{n-1}}$$

A scalar field has $n = 1$.

4.1.2 Rank 2 objects (to include dyadic tensor fields)

ADDR_MODEL: A general rank 2 tensor $t_{\alpha\beta}$ with components $(t_{0,0}, \dots, t_{m-1,n-1})$ is stored as:

$$\underbrace{\begin{array}{|c|c|c|c|c|c|c|c|}\hline t_{0,0} & t_{1,0} & \dots & t_{m-1,0} & t_{0,1} & t_{1,1} & \dots & t_{m-1,1}\\\hline\end{array}\cdots\begin{array}{|c|c|c|c|}\hline t_{0,n-1} & t_{1,n-1} & \dots & t_{m-1,n-1}\\\hline\end{array}}_{r_0}\cdots$$

$$\cdots\underbrace{\begin{array}{|c|c|c|c|c|c|c|c|}\hline t_{0,0} & t_{1,0} & \dots & t_{m-1,0} & t_{0,1} & t_{1,1} & \dots & t_{m-1,1}\\\hline\end{array}\cdots\begin{array}{|c|c|c|c|}\hline t_{0,n-1} & t_{1,n-1} & \dots & t_{m-1,n-1}\\\hline\end{array}}_{r_{n_s-1}}$$

Dyadic tensors, for example the gradient of a vector field $\partial_\alpha v_\beta$ in three dimensions, are stored in corresponding fashion with $m = 3$ and $\partial_\alpha = (\partial_x, \partial_y, \partial_z)$.

ADDR_MODEL_R: The Model R version is

$$\overbrace{\begin{array}{c}\begin{array}{|c|c|c|c|}\hline t_{0,0} & t_{0,0} & \dots & t_{0,0}\\\hline\end{array}\\ r_0 \quad r_1 \qquad r_{n_s-1}\end{array}}^{component_{0,0}}\overbrace{\begin{array}{c}\begin{array}{|c|c|c|c|}\hline t_{1,0} & t_{1,0} & \dots & t_{1,0}\\\hline\end{array}\\ r_0 \quad r_1 \qquad r_{n_s-1}\end{array}}^{component_{1,0}}\cdots\overbrace{\begin{array}{c}\begin{array}{|c|c|c|c|}\hline t_{m-1,n-1} & t_{m-1,n-1} & \dots & t_{m-1,n-1}\\\hline\end{array}\\ r_0 \quad r_1 \qquad r_{n_s-1}\end{array}}^{component_{m-1,n-1}}$$

4.1.3 Compressed rank 2 objects

A symmetric tensor $S_{\alpha\beta}$ in three dimensions has six independent components. It may be convenient to store this in compressed form as a rank 1 vector $(S_{xx}, S_{xy}, S_{xz}, S_{yy}, S_{yz}, S_{zz})$ to eliminate redundent storage.

A symmetric traceless rank 2 tensor — for example, the Landau-de Gennes liquid crystal order parameter $Q_{\alpha\beta}$ — has five independent components. This is stored as a rank 1 vector with five components $(Q_{xx}, Q_{xy}, Q_{xz}, Q_{yy}, Q_{yz})$ to eliminate redundant storage. API calls are provided to expand the compressed format to the full rank-2 representation $Q_{\alpha\beta}$ and, conversely, to compress the full representation to five components.

4.1.4 Rank 3 objects (to include triadic tensor fields)

The general rank 3 object $t_{\alpha\beta\gamma}$ with components $(t_{0,0,0}, \ldots, t_{m-1,n-1,p-1})$ is stored in a manner which generalises from the above, i.e., with the rightmost index running fastest. Diagrams are omitted, but Model and Model R storage patterns follow the same form as seen above.

A triadic tensor, for example the general second derivative of a vector field $\partial_\alpha \partial_\beta v_\gamma$ may be stored as a rank 3 object.

4.1.5 Compressed rank 3 objects

Symmetry properties may be used to reduce the storage requirement associated with rank 3 tensors. For example, the gradient of the liquid crystal order parameter $\partial_\gamma Q_{\alpha\beta}$ may be stored as a rank 2 object. The exact requirement will depend on the construction of the tensor.

4.1.6 LB distribution data

ADDR_MODEL: For the LB distributions, up to two distinct distributions can be accommodated to allow for a binary fluid implementation, although the usual situation is to have only one. The Model order for two N-velocity distributions f and g is:

$$\underbrace{\boxed{f_0}\boxed{f_1}\boxed{\ldots}\boxed{f_{N-1}}\boxed{g_0}\boxed{g_1}\boxed{\ldots}\boxed{g_{N-1}}}_{r_0}\cdots\underbrace{\boxed{f_0}\boxed{f_1}\boxed{\ldots}\boxed{f_{N-1}}\boxed{g_0}\boxed{g_1}\boxed{\ldots}\boxed{g_{N-1}}}_{r_{n_s=1}}$$

ADDR_MODEL_R: The Model R order is:

$$\overbrace{\overbrace{\underset{r_0}{\boxed{f_0}}\boxed{\cdots}\underset{r_{n_s-1}}{\boxed{f_0}}}^{f_0}\;\cdots\;\overbrace{\underset{r_0}{\boxed{f_{N-1}}}\boxed{\cdots}\underset{r_{n_s-1}}{\boxed{f_{N-1}}}}^{f_{N-1}}}^{f_i}\overbrace{\overbrace{\underset{r_0}{\boxed{g_0}}\boxed{\cdots}\underset{r_{n_s-1}}{\boxed{g_0}}}^{g_0}\;\cdots\;\overbrace{\underset{r_0}{\boxed{g_{N-1}}}\boxed{\cdots}\underset{r_{n_s-1}}{\boxed{g_{N-1}}}}^{g_{N-1}}}^{g_i}$$

For the single-distribution case, this is equivalent to the rank 1 vector field with $n = N$.

4.2 Generalised model for SIMD vectorisation

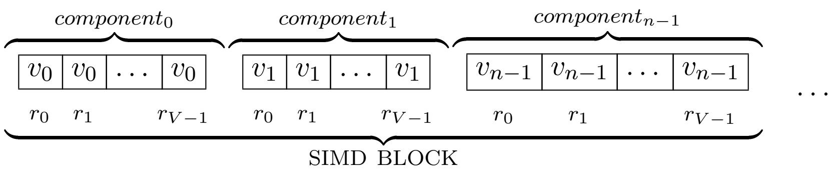
To promote parallelism at the instruction level, it is necessary to insure the innermost loop of any loop construct has an extent which is the vector length for the target architecture. The memory layout should therefore be adjusted accordingly. This means the MODEL format is generalised to a (rather clumsily named) array of structures of short vectors. The length of the short vectors is the SIMD vector length.

The outermost loop in this context is always to be the loop over lattice sites, which is adjusted according to the vector length; see Section 4.3 below for practical examples.

The Model R picture, which targets coalescene, can be viewed as naturally supporting SIMD vectorisation by virtue of allowing contiguous access to individual quantities as a function of lattice index. (If SIMD vectorisation is wanted at all on GPU architecture, it can be as a means to releive register pressure.) Model R therefore remains unaltered and we concentrate on the Model picture.

4.2.1 Rank 1 objects

VADDR_MODEL: The structure of short vectors is based on the SIMD vector length which we here denote V:



Subsequent SIMD blocks involve lattice sites $r_V \dots r_{2V-1}$, $r_{2V} \dots r_{3V-1}$, and so on. If the SIMD vector length is unity, this collapses to the standard Model picture shown in Section 4.1.1.

The generalisation of this approach to rank 2 and rank 3 objects follows the corresponding Model implementation.

4.3 Addressing: How to?

To provide a transparent interface for addressing vector fields, a single API is provided which implements the addressing order selected at compile time. This interface is always based on a fixed order of indices which may include the vector index of the innermost SIMD loop if present. This allows both vectorised and non-vectorised loops to be constructed in a consistent manner.

The usual model for constructing a loop involving all lattice sites (in this case without haloes) would be, in the absence of vectorisation:

```
for (ic = 1; ic <= nlocal[X]; ic++) {
  for (jc = 1; jc <= nlocal[Y]; jc++) {
    for (kc = 1; kc <= nlocal[Z]; kc++) {

      index = coords_index(ic, jc, kc);
      /* Perform operation per lattice site ... */
    }
  }
}
```

where final array indexing is based on the coordinate index and is specific to the object at hand. To allow transparent switching between Model and Model R versions, the indexing must be abstracted. As a concrete example, the following considers a rank 1 3-vector:

```
for (ic = 1; ic <= nlocal[X]; ic++) {
  for (jc = 1; jc <= nlocal[Y]; jc++) {
    for (kc = 1; kc <= nlocal[Z]; kc++) {

      index = coords_index(ic, jc, kc);
      for (ia = 0; ia < 3; ia++) {
        iref = addr_rank1(nsites, 3, index, ia);
        array[iref] = ...;
      }
    }
  }
}
```

Here, the function `addr_rank1()` performs the appropriate arithmetic to reference the correct 1-d array element in either address model. We note that this can be implemented to operate appropriately when the SIMD vector length is an arbitrary number.

An equivalent vectorised version may be constructed as follows:

```
for (n = 0; n < nsites; n += SIMDVL) {
  index = coords_indexv(n, nextra);   /* Proposal */
  for (ia = 0; ia < 3; ia++) {
    for (iv = 0; iv < SIMDVL; iv++) {
      iref = vaddr_rank1(nsites, 3, index, ia, iv);
      array[iref] = ...;
    }
  }
}
```

Note that a different function `vaddr_rank1()` with an additional argument which is the index of the vector loop is used. Note also that vectorisaed versions must take into account the extent that the kernel extends into the halo region (here `nextra`), which involves logic to avoid lattice sites which are not required (not shown in this example).

Note that, in practice, the loop over lattice sites is further abstracted in the code to accommodate thread-level parallelism via targetDP interface.

5 Lattice Boltzmann Hydrodynamics

We review here the lattice Boltzmann method applied to a simple Newtonian fluid with particular emphsis on the relevant implementation in *Ludwig*.

5.1 The Navier Stokes Equation

We seek to solve the isothermal Navier-Stokes equations which, often written in vector form, express mass conservation

$$\partial_t \rho + \boldsymbol{\nabla}.(\rho \mathbf{u}) = 0 \tag{1}$$

and the conservation of momentum

$$\partial_t(\rho \mathbf{u}) + \boldsymbol{\nabla}.(\rho \mathbf{u}\mathbf{u}) = -\boldsymbol{\nabla} p + \eta \nabla^2 \mathbf{u} + \zeta \boldsymbol{\nabla}(\boldsymbol{\nabla}.\mathbf{u}). \tag{2}$$

Equation 1 expresses the local rate of change of the density $\rho(\mathbf{r};t)$ as the divergence of the flux of mass associated with the velocity field $\mathbf{u}(\mathbf{r};t)$. Equation 2 expresses Newton's second law for momentum, where the terms on the right hand side represent the force on the fluid.

For this work, it is more convenient to rewrite these equations in tensor notation, where Cartesian coordinates x, y, z are represented by indices α and β, viz

$$\partial_t \rho + \nabla_\alpha(\rho u_\alpha) = 0 \tag{3}$$

and

$$\partial_t(\rho u_\alpha) + \nabla_\beta(\rho u_\alpha u_\beta) = -\nabla_\alpha p + \eta \nabla_\beta(u_\alpha \nabla_\beta + \nabla_\alpha u_\beta) + \zeta \nabla_\alpha(\nabla_\gamma u_\gamma). \tag{4}$$

Here, repeated Greek indices are understood to be summed over. The conservation law is seem better if the forcing terms of the right hand side are combined in the fluid stress $\Pi_{\alpha\beta}$ so that

$$\partial_t(\rho u_\alpha) + \nabla_\beta \Pi_{\alpha\beta} = 0. \tag{5}$$

In this case the expanded expression for the stress tensor is

$$\Pi_{\alpha\beta} = p\delta_{\alpha\beta} + \rho u_\alpha u_\beta + \eta \nabla_\alpha v_\beta + \zeta(\nabla_\gamma v_\gamma)\delta_{\alpha\beta} \tag{6}$$

where $\delta_{\alpha\beta}$ is that of Kroneker. Th Navier-Stokes equations in three dimensions have 10 degrees of freedom (or hydrodynamic modes) being ρ, three components of the mass flux ρu_α, and 6 independent modes from the (symmetric) stress tensor $\Pi_{\alpha\beta}$.

5.2 The Lattice Boltzmann Equation

The Navier-Stokes equation may be approximated in a discrete system by the lattice Boltzmann equation (LBE). A discrete density distribution function $f_i(\mathbf{r};t)$ at lattice points $\mathbf{r}$ and time t evolves according to

$$f_i(\mathbf{r} + \mathbf{c}_i \Delta t; t + \Delta t) = f_i(\mathbf{r};t) + \sum_j \mathcal{L}_{ij}\big(f_i(\mathbf{r};t) - f_i^{\mathrm{eq}}(\mathbf{r};t)\big) \tag{7}$$

where $\mathbf{c}_i$ is the discrete velocity basis and Δt is the discrete time step. The collision operator $\mathcal{L}_{ij}$ provides the mechanism to compute a discrete update from the non-equilibrium distribution $f_i(\mathbf{r};t) - f_i^{\mathrm{eq}}(\mathbf{r};t)$. Additional terms may be added to this equation to represent external body forces, thermal fluctuations, and so on. These additional terms are discussed in the following sections.

5.2.1 The distribution function and its moments

In lattice Boltzmann, the density and velocity of the continuum fluid are complemented by the distribution function $f_i(\mathbf{r};t)$ defined with reference to the discrete velocity space $c_{i\alpha}$. It is possible to relate the hydrodynamic quantities to the distribution function via its moments, that is

$$\rho(\mathbf{r};t) = \sum_i f_i(\mathbf{r};t), \quad \rho u_\alpha(\mathbf{r};t) = \sum_i f_i(\mathbf{r};t) c_{i\alpha}, \quad \Pi_{\alpha\beta}(\mathbf{r};t) = \sum_i f_i(\mathbf{r};t) c_{i\alpha} c_{i\beta}. \tag{8}$$

Here, the index of the summation is over the number of discrete velocities used as the basis, a number which will be denoted N_{vel}. For example, in three dimensions N_{vel} is often 19 and the basis is referred to as D3Q19.

The number of moments, or modes, supported by a velocity set is exactly N_{vel}, and these can be written in general as

$$M^a(\mathbf{r};t) = \sum_i m_i^a f_i(\mathbf{r};t), \tag{9}$$

where the m_i are the eigenvectors of the collision matrix in the LBE. For example, in the case of the density, all the $m_i^a = 1$ and the mode M^a is the density $\rho = \sum_i f_i$. Note that the number of modes supported by a given basis will generally excceed the number of hydrodynamic modes; the excess modes have no direct physical interpretation and are variously referred to as non-hydrodynamic, kinetic, or ghost, modes. The ghost modes take no part in bulk hydrodynamics, but may become important in other contexts, such as thermal fluctuations and near boundaries. The distribution function can be related to the modes $M^a(\mathbf{r};t)$ via

$$f_i(\mathbf{r};t) = w_i \sum_a m_i^a N^a M^a(\mathbf{r};t). \tag{10}$$

In this equation, w_i are the standard LB weights appearing in the equilibrium distribution function, while the N^a are a per-mode normalising factor uniquely determined by the orthogonality condition

$$N^a \sum_i w_i m_i^a m_i^b = \delta_{ab}. \tag{11}$$

Writing the basis this way has the advantage that the equilibrium distribution projects directly into the hydrodynamic modes only. Putting it another way, we may write

$$f_i^{\mathrm{eq}} = w_i\big(\rho + \rho c_{i\alpha} u_\alpha / c_s^2 + (c_{i\alpha} c_{i\beta} - c_s^2 \delta_{\alpha\beta})(\Pi_{\alpha\beta}^{\mathrm{eq}} - p\delta_{\alpha\beta})/2c_s^4\big) \tag{12}$$

where only (equilibrium) hydrodynamic quantities appear on the right hand side.

5.2.2 Collision and relaxation times

5.3 Model Basis Descriptions

5.3.1 D2Q9

The D2Q9 model in two dimensions consists one zero vector (0,0), four vectors of length unity being $(\pm1, 0)$ and $(0, \pm1)$, and four vectors of length $\sqrt{2}$ being $(\pm1, \pm1)$. The eigenvectors of the collision matrix, with associated weights and normalisers are shown in Table 1. In two dimensions there are six hydrodynamic modes and a total of three kinetic modes, or ghost modes.

M^a	p	m_i^a									N^a	
ρ	-	1	1	1	1	1	1	1	1	1	1	**1**
ρc_{ix}	-	0	1	1	1	0	0	-1	1	-1	3	c_{ix}
ρc_{iy}	-	0	1	0	-1	1	-1	1	0	-1	3	c_{iy}
Q_{xx}	1/3	-1	2	2	2	-1	-1	2	2	2	9/2	$c_{ix}c_{ix} - c_s^2$
Q_{xy}	-	0	1	0	-1	0	0	-1	0	1	9	$c_{ix}c_{iy}$
Q_{yy}	1/3	-1	2	-1	2	2	2	2	-1	2	9/2	$c_{iy}c_{iy} - c_s^2$
χ^1	-	1	4	-2	4	-2	-2	4	-2	4	1/4	χ^1
J_{ix}	-	0	4	-2	4	0	0	-4	-2	-4	3/8	$\chi^1 \rho c_{ix}$
J_{iy}	-	0	4	0	-4	-2	2	4	0	-4	3/8	$\chi^1 \rho c_{iy}$
w_i	1/36	16	1	4	1	4	4	1	4	1		w_i

Table 1: Table showing the details of the basis used for the D2Q9 model in two dimensions. The nine modes M^a include six hydrodynamic modes, one scalar kinetic mode χ^1, and one vector kinetic mode $J_{i\alpha}$. The weights in the equilibrium distribution function are w_i and the normaliser for each mode is N^a. The eigenvectors of the collision matrix are the columns of the transformation matrix m_i^a. The prefactor p (where present) multiplies all the elements to the right in that row.

5.3.2 D3Q15

The D3Q15 model in three dimensions consists of a set of vectors: one zero vector $(0, 0, 0)$, six vectors of length unity being $(\pm 1, 0, 0)$ cyclically permuted, and 8 vectors of length $\sqrt{3}$ being $(\pm 1, \pm 1, \pm 1)$. The eigenvalues and eigenvectors of the collision matrix used for D3Q15 are given in Table 2.

5.3.3 D3Q19

The D3Q19 model in three dimensions is constructed with velocities: one zero vector $(0, 0, 0)$, three vectors of length unity being $(\pm 1, 0, 0)$ cyclically permuted, and twelve vectors of length $\sqrt{2}$ being $(\pm 1, \pm 1, 0)$ cyclically permuted. The details of the D3Q19 model are set out in Table 3.

5.4 Fluctuating LBE

It is possible [3] to simulate fluctuating hydrodynamics for an isothermal fluid via the inclusion of a fluctuating stress $\sigma_{\alpha\beta}$:

$$\Pi_{\alpha\beta} = p\delta_{\alpha\beta} + \rho u_\alpha u_\beta + \eta_{\alpha\beta\gamma\delta}\nabla_\gamma u_\delta + \sigma_{\alpha\beta}. \tag{13}$$

The fluctuation-dissipation theorem relates the magnitude of this random stress to the isothermal temperature and the viscosity.

In the LBE, this translates to the addition of a random contribution ξ_i to the distribution at the collision stage, so that

$$\ldots + \xi_i. \tag{14}$$

For the conserved modes $\xi_i = 0$. For all the non-conserved modes, i.e., those with dissipation, the fluctuating part may be written

$$\xi_i(\mathbf{r}; t) = w_i m_i^a \hat{\xi}^a(\mathbf{r}; t) N^a \tag{15}$$

M^a	p	m_i^a															N^a	
ρ	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**
ρc_{ix}	-	0	1	-1	0	0	0	0	1	-1	1	-1	1	-1	1	-1	3	c_{ix}
ρc_{iy}	-	0	0	0	1	-1	0	0	1	1	-1	-1	1	1	-1	-1	3	c_{iy}
ρc_{iz}	-	0	0	0	0	0	1	-1	1	1	1	1	-1	-1	-1	-1	3	c_{iz}
Q_{xx}	1/3	-1	2	2	-1	-1	-1	-1	2	2	2	2	2	2	2	2	9/2	$c_{ix}c_{ix} - c_s^2$
Q_{yy}	1/3	-1	-1	-1	2	2	-1	-1	2	2	2	2	2	2	2	2	9/2	$c_{iy}c_{iy} - c_s^2$
Q_{zz}	1/3	-1	-1	-1	-1	-1	2	2	2	2	2	2	2	2	2	2	9/2	$c_{iz}c_{iz} - c_s^2$
Q_{xy}	-	0	0	0	0	0	0	0	1	-1	-1	1	1	-1	-1	1	9	$c_{ix}c_{iy}$
Q_{yz}	-	0	0	0	0	0	0	0	1	1	-1	-1	-1	-1	1	1	9	$c_{iy}c_{iz}$
Q_{zx}	-	0	0	0	0	0	0	0	1	-1	1	-1	-1	1	-1	1	9	$c_{iz}c_{ix}$
χ^1	-	-2	1	1	1	1	1	1	-2	-2	-2	-2	-2	-2	-2	-2	1/2	χ^1
J_{ix}	-	0	1	-1	0	0	0	0	-2	2	-2	2	-2	2	-2	2	3/2	$\chi^1 \rho c_{ix}$
J_{iy}	-	0	0	0	1	-1	0	0	-2	-2	2	2	-2	-2	2	2	3/2	$\chi^1 \rho c_{iy}$
J_{iz}	-	0	0	0	0	0	1	-1	-2	-2	-2	-2	2	2	2	2	3/2	$\chi^1 \rho c_{iz}$
χ^3	-	0	0	0	0	0	0	0	1	-1	-1	1	-1	1	1	-1	9	$c_{ix}c_{iy}c_{iz}$
w_i	1/72	16	8	8	8	8	8	8	1	1	1	1	1	1	1	1		w_i

Table 2: Table showing the details of the basis used for the D3Q15 model in three dimensions. The fifteen modes M^a include two scalar kinetic modes χ^1 and χ^3, and one vector kinetic mode $J_{i\alpha}$. The weights in the equilibrium distribution are w_i and the normaliser for each mode is N^a. The eigenvectors of the collision matrix are the columns of the transformation matrix m_i^a. The prefactor p simply multiplies all elements of m_i^a in that row as a convenience.

where $\hat{\xi}^a$ is a noise termwhich has a variance determined by the relaxation time for given mode

$$\left\langle \hat{\xi}^a \hat{\xi}^b \right\rangle = \frac{\tau_a + \tau_b + 1}{\tau_a \tau_b} \left\langle \delta M^a \delta M^b \right\rangle . \tag{16}$$

5.4.1 Fluctuating stress

For the stress, the random contribution to the distributions is

$$\xi_i = w_i \frac{Q_{i\alpha\beta} \hat{\sigma}_{\alpha\beta}}{4c_s^2} \tag{17}$$

where $\hat{\sigma}_{\alpha\beta}$ is a symmtric matrix of random variates drawn from a Gaussian distribution with variance given by equation 16. In the case that the shear and bulk viscosities are the same, i.e., there is a single relaxation time, then the variances of the six independent components of the matrix are given by

$$\langle \hat{\sigma}_{\alpha\beta} \hat{\sigma}_{\mu\nu} \rangle = \frac{2\tau + 1}{\tau^2} (\delta_{\alpha\mu}\delta_{\beta\nu} + \delta_{\alpha\nu}\delta_{\beta\mu}). \tag{18}$$

5.5 Hydrodynamic Boundary Conditions

5.5.1 Bounce-Back on Links

A very general method for the representation of solid objects within the LB approach was put forward by Ladd [24, 25]. Solid objects (of any shape) are defined by a boundary

M^a	m_i^a																			N^a
ρ	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ρc_{ix}	0	1	-1	0	0	0	0	1	1	-1	-1	1	1	-1	-1	0	0	0	0	3
ρc_{iy}	0	0	0	1	-1	0	0	1	-1	1	-1	0	0	0	0	1	1	-1	-1	3
ρc_{iz}	0	0	0	0	0	1	-1	0	0	0	0	1	-1	1	-1	1	-1	1	-1	3
Q_{ixx}	-1	2	2	-1	-1	-1	-1	2	2	2	2	2	2	2	2	-1	-1	-1	-1	9/2
Q_{iyy}	-1	-1	-1	2	2	-1	-1	2	2	2	2	-1	-1	-1	-1	2	2	2	2	9/2
Q_{izz}	-1	-1	-1	-1	-1	2	2	-1	-1	-1	-1	2	2	2	2	2	2	2	2	9/2
Q_{ixy}	0	0	0	0	0	0	0	1	-1	-1	1	0	0	0	0	0	0	0	0	9
Q_{ixz}	0	0	0	0	0	0	0	0	0	0	0	1	-1	-1	1	0	0	0	0	9
Q_{iyz}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	-1	1	9
χ^1	0	1	1	1	1	-2	-2	-2	-2	-2	-2	1	1	1	1	1	1	1	1	3/4
$\chi^1 \rho c_{ix}$	0	1	-1	0	0	0	0	-2	-2	2	2	1	1	-1	-1	0	0	0	0	3/2
$\chi^1 \rho c_{iy}$	0	0	0	1	-1	0	0	-2	2	-2	2	0	0	0	0	1	1	-1	-1	3/2
$\chi^1 \rho c_{iz}$	0	0	0	0	0	-2	2	0	0	0	0	1	-1	1	-1	1	-1	1	-1	3/2
χ^2	0	1	1	-1	-1	0	0	0	0	0	0	-1	-1	-1	-1	1	1	1	1	9/4
$\chi^2 \rho c_{ix}$	0	1	-1	0	0	0	0	0	0	0	0	-1	-1	1	1	0	0	0	0	9/2
$\chi^2 \rho c_{iy}$	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	1	1	-1	-1	9/2
$\chi^2 \rho c_{iz}$	0	0	0	0	0	0	0	0	0	0	0	-1	1	-1	1	1	-1	1	-1	9/2
χ^3	1	-2	-2	-2	-2	-2	-2	1	1	1	1	1	1	1	1	1	1	1	1	1/2
w_i	12	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	

Table 3: Table showing the details of the basis used for the D3Q19 model in three dimensions. The nineteen modes M^a include ten hydrodynamic modes, three scalr kinetic modes χ^1, χ^2, and χ^3; there are also two vector kinetic modes $\chi^1 \rho c_{i\alpha}$ and $\chi^2 \rho c_{i\alpha}$. The weights in the equilibrium distribution function are w_i, and the normaliser for each mode is N^a. The eigenvectors of the collision matrix are the columns of the transformation matrix m_i^a.

surface which intersects some of the velocity vectors $\mathbf{c}_i$ joining lattice nodes. Sites inside are designated solid, while sites outside remain fluid. The correct boundary condition is defined by identifying *links* between fluid and solid sites, which allows those elements of the distribution which would cross the boundary at the propagation step to be "bounced-back" into the fluid. This bounce-back on links is an efficient method to obtain the correct hydrodynamic interaction between solid and fluid.

5.5.2 Fixed objects

5.5.3 Moving objects

6 The Input File

6.1 General

By default, the run time expects to find user input in a file `input` in the current working directory. If a different file name is required, its name should be provided as the sole command line argument, e.g.,

```
./Ludwig.exe input_file_name
```

If the input file is not located in the current working directory the code will terminate immediately with an error message.

When an input file is located, its content is read by a single MPI task, and its contents then broadcast to all MPI relevant tasks. The format of the file is plain ASCII text, and its contents are parsed on a line by line basis. Lines may contain the following:

- comments introduced by `#`.
- key value pairs separated by white space.

Blank lines are treated as comments. The behaviour of the code is determined by a set of key value pairs. Any given key may appear only once in the input file; unused key value pairs are not reported. If the key value pairs are not correctly formed, the code will terminate with an error message and indicate the offending input line.

Key value pairs may be present in the input file, but have no effect for any given run: they are merely ignored. Relevant control parameters for given input are reported in the standard output.

6.1.1 Key value pairs

Key value pairs are made up of a key — an alphanumeric string with no white space — and corresponding value following white space. Values may take on the follow forms:

```
key_string         value_string

key_integer_scalar 1
key_integer_vector 1_2_3

key_double_scalar  1.0
key_double_vector  1.0_2.0_3.0
```

Values which are strings should contain no white space. Scalar parameters may be integer values, or floating point values with a decimal point (scientific notation is also allowed). Vector parameters are introduced by a set of three values (to be interpreted as x, y, z components of the vector in Cartesian coordinates) separated by an underscore. The identity of the key will specify what type of value is expected. Keys and (string) values are case sensitive.

Most keys have an associated default value which will be used if the key is not present. Some keys must be specified: an error will occur if they are missing. The remainder of this part of the guide details the various choices for key value pairs, along with any default values, and any relevant constraints.

6.2 The Free Energy

The choice of free energy is determined as follows:

```
free_energy  none
```

The default value is `none`, i.e., a simple Newtonian fluid is used. Possible values of the `free_energy` key are:

```
#  none                   Newtonian fluid [DEFAULT]
#  symmetric              Symmetric binary fluid (finite difference)
#  symmetric_lb           Symmetric binary fluid (two distributions)
#  brazovskii             Brazovskii smectics
#  surfactant             Surfactants
#  polar_active           Polar active gels
#  lc_blue_phase          Liquid crystal (nematics, cholesterics, BPs)
#  lc_droplet             Liquid crystal emulsions
#  fe_electro             Single fluid electrokinetics
#  fe_electro_symmetric   Binary fluid electrokinetics
```

The choice of free energy will control automatically a number of factors related to choice of order parameter, the degree of parallel communication required, and so on. Each free energy has a number of associated parameters discussed in the following sections.

Details of general (Newtonian) fluid parameters, such as viscosity, are discussed in Section 6.4.

6.2.1 Symmetric Binary Fluids

We recall that the free energy density is, as a function of compositional order ϕ:

$$\tfrac{1}{2}A\phi^2 + \tfrac{1}{4}B\phi^4 + \tfrac{1}{2}\kappa(\nabla\phi)^2.$$

Parameters are introduced by (with default values):

```
free_energy symmetric
A           -0.0625                        # Default: -0.003125
B           +0.0625                        # Default: +0.003125
K           +0.04                          # Default: +0.002
```

Common usage has $A < 0$ and $B = -A$ so that $\phi^\star = \pm 1$. The parameter κ (key `K`) controls the interfacial energy penalty and is usually positive.

6.2.2 Brazovskii smectics

The free energy density is:

$$\tfrac{1}{2}A\phi^2 + \tfrac{1}{4}B\phi^4 + \tfrac{1}{2}\kappa(\nabla\phi)^2 + \tfrac{1}{2}C(\nabla^2\phi)^2$$

Parameters are introduced via the keys:

```
free_energy brazovskii
A            -0.0005                       # Default: 0.0
B            +0.0005                       # Default: 0.0
K            -0.0006                       # Default: 0.0
C            +0.00076                      # Default: 0.0
```

For $A < 0$, phase separation occurs with a result depending on κ: one gets two symmetric phases for $\kappa > 0$ (cf. the symmetric case) or a lamellar phase for $\kappa < 0$. Typically, $B = -A$ and the parameter in the highest derivative $C > 0$.

6.2.3 Surfactants

The surfactant free energy should not be used at the present time.

6.2.4 Polar active gels

The free energy density is a function of vector order parameter P_α:

$$\tfrac{1}{2}AP_\alpha P_\alpha + \tfrac{1}{4}B(P_\alpha P_\alpha)^2 + \tfrac{1}{2}\kappa(\partial_\alpha P_\beta)^2$$

There are no default parameters:

```
free_energy      polar_active
polar_active_a  -0.1                    # Default: 0.0
polar_active_b  +0.1                    # Default: 0.0
polar_active_k   0.01                   # Default: 0.0
```

It is usual to choose $B > 0$, in which case $A > 0$ gives an isotropic phase, whereas $A < 0$ gives a polar nematic phase. The elastic constant κ (key `polar_active_k`) is positive.

6.2.5 Liquid crystal

The free energy density is a function of tensor order parameter $Q_{\alpha\beta}$:

$$\begin{aligned}\tfrac{1}{2}A_0(1-\gamma/3)Q_{\alpha\beta}^2 - \tfrac{1}{3}A_0\gamma Q_{\alpha\beta}Q_{\beta\delta}Q_{\delta\alpha} + \tfrac{1}{4}A_0\gamma(Q_{\alpha\beta}^2)^2 \\ +\tfrac{1}{2}\Big(\kappa_0(\epsilon_{\alpha\delta\sigma}\partial_\delta Q_{\sigma\beta} + 2q_0 Q_{\alpha\beta})^2 + \kappa_1(\partial_\alpha Q_{\alpha\beta})^2\Big)\end{aligned}$$

The corresponding `free_energy` value, despite its name, is suitable for nematics and cholesterics, and not just blue phases:

```
free_energy     lc_blue_phase
lc_a0           0.01                   # Deafult: 0.0
lc_gamma        3.0                    # Default: 0.0
lc_q0           0.19635                # Default: 0.0
lc_kappa0       0.00648456             # Default: 0.0
lc_kappa1       0.00648456             # Default: 0.0
```

The bulk free energy parameter A_0 is positive and controls the energy scale (key `lc_a0`); γ is positive and influences the position in the phase diagram relative to the isotropic/nematic transition (key `lc_gamma`). The two elastic constants must be equal, i.e., we enforce the single elastic constant approximation (boths keys `lc_kappa0` and `lc_kappa1` must be specified).

Other important parameters in the liquid crystal picture are:

```
lc_xi           0.7                     # Default: 0.0
lc_Gamma        0.5                     # Default: 0.0
lc_active_zeta  0.0                     # Default: 0.0
```

The first is ξ (key `lc_xi`) is the effective molecular aspect ratio and should be in the range $0 < \xi < 1$. The rotational diffusion constant is Γ (key `lc_Gamma`; not to be confused with `lc_gamma`). The (optional) apolar activity parameter is ζ (key `lc_active_zeta`).

6.2.6 Liquid crystal emulsion

This an interaction free energy which combines the symmetric and liquid crystal free energies. The liquid crystal free energy constant γ becomes a function of composition via $\gamma(\phi) = \gamma_0 + \delta(1+\phi)$, and a coupling term is added to the free energy density:

$$W Q_{\alpha\beta} \partial_\alpha \phi \partial_\beta \phi.$$

Typically, we might choose γ_0 and δ so that $\gamma(-\phi^\star) < 2.7$ and the $-\phi^\star$ phase is isotropic, while $\gamma(+\phi^\star) > 2.7$ and the $+\phi^\star$ phase is ordered (nematic, cholesteric, or blue phase). Experience suggests that a suitable choice is $\gamma_0 = 2.5$ and $\delta = 0.25$.

For anchoring constant $W > 0$, the liquid crystal anchoring at the interface is planar, while for $W < 0$ the anchoring is normal. This is set via key `lc_droplet_W`.

Relevant keys (with default values) are:

```
free_energy          lc_droplet

A                    -0.0625
B                    +0.0625
K                    +0.053

lc_a0                 0.1
lc_q0                 0.19635
lc_kappa0             0.007
lc_kappa1             0.007

lc_droplet_gamma      2.586              # Default: 0.0
lc_droplet_delta      0.25               # Default: 0.0
lc_droplet_W         -0.05               # Default: 0.0
```

Note that key `lc_gamma` is not set in this case.

6.3 System Parameters

Basic parameters controlling the number of time steps and the system size are:

```
N_start     0                            # Default: 0
N_cycles    100                          # Default: 0
size        128_128_1                    # Default: 64_64_64
```

A typical simulation will start from time zero (key `N_start`) and run for a certain number of time steps (key `N_cycles`). The system size (key `size`) specifies the total number of lattice sites in each dimension. If a two-dimensional system is required, the extent in the z-direction must be set to unity, as in the above example.

If a restart from a previous run is required, the choice of parameters may be as follows:

```
N_start     100
N_cycles    400
```

This will restart from data previously saved at time step 100, and run a further 400 cycles, i.e., to time step 500.

6.3.1 Parallel decomposition

In parallel, the domain decompostion is closely related to the system size, and is specified as follows:

```
size       64_64_64
grid       4_2_1
```

The `grid` key specifies the number of MPI tasks required in each coordinate direction. In the above example, the decomposition is into 4 in the x-direction, into 2 in the y-direction, while the z-direction is not decomposed. In this example, the local domain size per MPI task would then be $16 \times 32 \times 64$. The total number of MPI tasks available must match the total implied by `grid` (8 in the example).

The `grid` specifications must exactly divide the system size; if no decomposition is possible, the code will terminate with an error message. If the requested decomposition is not valid, or `grid` is omitted, the code will try to supply a decomposition based on the number of MPI tasks available and `MPI_Dims_create()`; this may be implementation dependent.

6.4 Fluid Parameters

Control parameters for a Newtonian fluid include:

```
fluid_rho0              1.0
viscosity               0.166666666666666
viscosity_bulk          0.166666666666666
isothermal_fluctuations off
temperature             0.0
```

The mean fluid density is ρ_0 (key `fluid_rho0`) which defaults to unity in lattice units; it is not usually necessary to change this. The shear viscosity is `viscosity` and as default value 1/6 to correspond to unit relaxation time in the lattice Boltzmann picture. Reasonable values of the shear viscosity are $0.2 > \eta > 0.0001$ in lattice units. Higher values move further into the over-relaxation region, and can result in poor behaviour. Lower values increase the Reynolds number and tend to cause problems with stability. The bulk viscosity has a default value which is equal to whatever shear viscosity has been selected. Higher values of the bulk viscosity may be set independently and can help to suppress large deviations from incompressibility and maintain numerical stability in certain situations.

If fluctuating hydrodynamics is wanted, set the value of `isothermal_fluctualtions` to `on`. The associated temperature is in lattice units: reasonable values (at $\rho_0 = 1$) are $0 < kT < 0.0001$. If the temperature is too high, local velocities will rapidly exceed the Mach number constraint and the simulation will be unstable.

7 Binary Fluids

7.1 The Cahn-Hilliard Equation

7.2 The Choice of Free Energy

7.3 A Second Distribution Function

7.4 The Collision

7.5 Bounce-Back on Links

7.6 Upwind Advection Schemes

The solution to the Cahn-Hilliard equation for the order parameter

$$\partial_t \phi + \partial_\alpha (u_\alpha \phi + M \partial_\alpha \mu) = 0 \tag{19}$$

assumes that the valocity field u_α is known, along with the order parameter mobility M. Adopting a divergence form ensures that any finite difference scheme conserves the total order parameter in the system.

Considering just the advective part of 19, a finite difference approach boils down to finding an interpolation of $\phi(\mathbf{r})$ to the faces of a control volume surrounding the lattice site at $\mathbf{r}$. So, in one dimension we have

$$\phi_i^{n+1} = \phi_i^n + \frac{\Delta t}{\Delta x}(u_w \phi_w - u_e \phi_e) \tag{20}$$

where subscripts w and e refer to compass directions with index i incresing eastward. A first order upwind scheme approximates the interfacial value ϕ_w depending on the direction of the velocity at the face, viz:

$$\phi_w = \begin{cases} \phi_i & u_w < 0, \\ \phi_{i-1} & u_w >= 0. \end{cases} \tag{21}$$

This choice is conditionally stable for the Euler forward time step of Eq. (20), but highly dissipative.

Better accuracy requires a higher-order approximation to the interfacial value of ϕ.

7.6.1 Uniformly third order scheme

We follow Leonard et al. [**?**] in adopting a uniformly third order approximation dependent upon both the normal and tangential interfacial Courant numbers. It is extended here to three dimensions.

Consider the two dimensional problem of Figure **??**.

The flux at face W is then

$$
\begin{aligned}
c_x\Big\{\; & 1/2(\phi_c+\phi_w) - 1/2c_x(\phi_c-\phi_w) - 1/6(1-c_x^2)(\phi_c - 2\phi_w + \phi_{ww}) \\
& -1/2c_y(\phi_w - \phi_{sw}) \\
& -c_y(1/4 - 1/3c_x)(\phi_c - \phi_w - \phi_s + \phi_{sw}) - c_y(1/4 - 1/6c_y)(\phi_{nw} - 2\phi_w + \phi_{sw}) \\
& -1/2c_z(\phi_w - \phi_{wd}) \\
& -c_z(1/4 - 1/3c_x)(\phi_c - \phi_w - \phi_{cd} + \phi_{wd}) - c_z(1/4 - 1/6c_y)(\phi_{wu} - 2\phi_w + \phi_{wd}) \\
& +c_yc_z\big[1/3(\phi_w - \phi_{wd}) - 1/3(\phi_{sw} - \phi_{swd}) \\
& +(1/6 - 1/4c_x)(\phi_c - \phi_w - \phi_s + \phi_{sw} - (\phi_{cd} - \phi_{wd} - \phi_{sd} + \phi_{swd})) \\
& +(1/6 - 1/8c_y)(\phi_{nw} - 2\phi_w + \phi_{sw} - (\phi_{nwd} - 2\phi_{wd} + \phi_{swd})) \\
& +(1/6 - 1/8c_z)(\phi_{wu} - 2\phi_w + \phi_{wd} - (\phi_{swu} - 2\phi_{sw} + \phi_{swd}))\big]\Big\}
\end{aligned}
$$

7.6.2 Gory details

The flux-integral method of Leonard et al. assumes that within a given cell, the order parameter is piecewise quadratic. So, for cell W we have

$$\phi(\zeta,\eta,\theta) = \phi_W - \tfrac{1}{24}(\phi_C + \phi_{SW} + \phi_{WW} + \phi_{NW} + \phi_{WD} + \phi_{WU} - 6\phi_W) \tag{22}$$
$$+\tfrac{1}{2}(\phi_C - \phi_{WW})\zeta + \tfrac{1}{2}(\phi_C - 2\phi_W + \phi_{WW})\zeta^2 \tag{23}$$
$$+\tfrac{1}{2}(\phi_{NW} - \phi_{SW})\eta + \tfrac{1}{2}(\phi_{NW} - 2\phi_W + \phi_{SW})\eta^2 \tag{24}$$
$$+\tfrac{1}{2}(\phi_{WU} - \phi_{WD})\theta + \tfrac{1}{2}(\phi_{WU} - 2\phi_W + \phi_{WD})\theta^2 \tag{25}$$

7.7 Lattice kinetic equation viewed as finite difference

For binary fluid problems the second distribution $g_i(\mathbf{x};t)$, representing the composition, obeys the evolution equation

$$g_i(\mathbf{x};t+\Delta t) = g_i^\star(\mathbf{x} - \mathbf{c}_i\Delta t;t) \tag{26}$$

where the star indicates the post-collision distribution. Here we consider the distribution to be set by requiring the first moment

$$j_\alpha = \sum_i g_i^\star(\mathbf{x};t)c_{i\alpha} = \phi u_\alpha \tag{27}$$

and the second moment as

$$\Phi_{\alpha\beta} = \sum_i g_i^\star(\mathbf{x};t)c_{i\alpha}c_{i\beta} = \phi u_\alpha u_\beta + 2M\mu(\mathbf{x};t)\delta_{\alpha\beta}. \tag{28}$$

Here the mobility M enters through the terms related to the chemical potential $\mu(\mathbf{x};t)$. The distribution is then set using the reprojection

$$g_i^\star(\mathbf{x};t) = \delta_{i0}\phi + w_i(j_\alpha u_{i\alpha}/c_s^2 + \Phi_{\alpha\beta}Q_{i\alpha\beta}/2c_s^4), \tag{29}$$

where the δ_{i0} moves ϕ mostly into the non-propagating distribution.

For a uniform velocity field $u_\alpha = (u_x, u_y, u_z)$ it is possible to expand **??** in a finite difference form which includes three parts: the advective part in ϕ related to the first moment, a dissipative contribution in ϕ related to the $\phi u_\alpha u_\beta$ term in the second moment, and a part related to the diffusion of the chemical potential.

7.7.1 Advective terms

Combining **??** and 27 and taking the sum over the distributions we have

$$\sum_i g_i(\mathbf{x};t) = \sum_i \Big(g_0^\star(\mathbf{x};t) + (1/c_s^2)w_i\phi(\mathbf{x}-\mathbf{c}_i\Delta t;t)u_\alpha c_{i\alpha}\Big). \tag{30}$$

Introducing a finite finiterence notation $\phi^n_{ijk} = \sum_{i'} g^\star_{i'}(\mathbf{x};t)$ where the indices i,j,k represent the spatial discretistation, and the superscript n represents the discrete time level we have

$$\phi^{n+1}_{ijk} = \phi^n_{ijk} - u_x w_1/c_s^2\Big\{\phi^n_{i+1jk} - \phi^n_{i-1jk} \tag{31}$$

$$+1/2(\phi^n_{i+1j+1k} - \phi^n_{i-1j-1k} + \phi^n_{i+1j-1k} - \phi^n_{i-1j+1k} + \phi^n_{i+1jk+1} - \phi^n_{i-1jk-1} + \phi^n_{i+1jk-1} - \phi^n_{i-1jk-+})\Big\} \tag{32}$$

$$+u_y w_1/c_s^2\Big\{\phi^n_{ij+1k} - \phi^n_{ij-1k} \tag{33}$$

$$+1/2(\phi^n_{i+1j+1k} - \phi^n_{i-1j-1k} + \phi^n_{i-1j+1k} - \phi^n_{i+1j-1k} + \phi^n_{ij+1k+1} - \phi^n_{ij-1k-1} + \phi^n_{ij+1k-1} - \phi^n_{ij-1k+1})\Big\} \tag{34}$$

$$+u_z w_1/c_s^2\Big\{\phi^n_{ijk+1} - \phi^n_{ijk-1} \tag{35}$$

$$+1/2(\phi^n_{i+1jk+1} - \phi^n_{i-1jk-1} + \phi^n_{i-1jk+1} - \phi^n_{i+1jk-1} + \phi^n_{ij+1k+1} - \phi^n_{ij-1k-1} + \phi^n_{ij-1k+1} - \phi^n_{ij+1k-1})\Big\}. \tag{36}$$

We can regcognise here second order centred difference approximations to the first derivative in ϕ in the coordinate direction in the diagonal directions (cf [**?**] with 'isotropy correction factor' $\beta = 1/2$ in three dimensions).

There are a number of points to note about this. First, there are terms in $\sum_i g_0^\star$ which are not included above, but are dealt with below. Second, while the spatial discretisation looks ok, this is forward in time, which would classically be undesirable. Dissipation is required to prevent dispersive errors dominating. (Although with general flow field, the coefficients of the finite difference form are altered in ways which could provide some upwind bias.)

7.7.2 Dissipative terms

The dissipative terms from the $\phi u_\alpha u_\beta$ term in the second moment give

$$\sum_{i'} g_{i'}(\mathbf{x};t+\Delta t) = 1/2c_s^4 \sum_{i'} w_i\phi(\mathbf{x}-\mathbf{c}_i\Delta t;t)u_\alpha u_\beta Q_{i'\alpha\beta}. \tag{37}$$

With a little effort this may be expanded to give and evolution equation with the following difusive tendency terms:

$$\begin{aligned}
w_1/2c_s^4\Big\{&(u_x^2 - c_s^2u^2)(\phi_{i+1jk} - 2\phi_{ijk} + \phi_{i-1jk}) && (38)\\
&+(u_y^2 - c_s^2u^2)(\phi_{ij+1k} - 2\phi_{ijk} + \phi_{ij-1k}) && (39)\\
&+(u_z^2 - c_s^2u^2)(\phi_{ijk+1} - 2\phi_{ijk} + \phi_{ijk-1})\Big\} && (40)\\
+w_2/2c_s^4\Big\{&\big[(u_x^2 + u_y^2) - c_s^2u^2\big](\phi_{i+1j+1k} - 2\phi_{ijk} + \phi_{i-1j-1k}) && (41)\\
&+\big[(u_x^2 - u_y^2) - c_s^2u^2\big](\phi_{i+1j-1k} - 2\phi_{ijk} + \phi_{i-1j+1k}) && (42)\\
&+\big[(u_x^2 + u_z^2) - c_s^2u^2\big](\phi_{i-1jk-1} - 2\phi_{ijk} + \phi_{i+1jk+1}) && (43)\\
&+\big[(u_x^2 - u_z^2) - c_s^2u^2\big](\phi_{i+1jk-1} - 2\phi_{ijk} + \phi_{i-1jk+1}) && (44)\\
&+\big[(u_y^2 + u_z^2) - c_s^2u^2\big](\phi_{ij-1k-1} - 2\phi_{ijk} + \phi_{ij+1k+1}) && (45)\\
&+\big[(u_y^2 - u_z^2) - c_s^2u^2\big](\phi_{ij+1k-1} - 2\phi_{ijk} + \phi_{ij-1k+1})\Big\}. && (46)
\end{aligned}$$

It can be see that the coefficients of these diffusive terms is related to the aspect ratio of the flow, and that the coefficents of given terms could become negative.

7.7.3 Chemical potential term

Finally, the terms in the chemical potential give

$$\sum_{i'} g(\mathbf{x}; t + \Delta t) = \sum_{i'} w_i 2M\mu(\mathbf{x} - \mathbf{c}_\Delta t)\delta_{\alpha\beta}Q_{i\alpha\beta}. \tag{47}$$

Noting that the contraction $\delta_{\alpha\beta}Q_{i\alpha\beta} = -1, 0, 1$ for velocities with weights w_0, w_1, w_2 respectively, we get

$$\begin{aligned}
\phi_{ijk}^{n+1} = \phi_{ijk}^n - w_0 2M u_{ijk}/2c_s^4 + 2Mw_2/2c_s^4\Big\{&\mu_{i+1j+1k} + \mu_{i-1j-1k} + \mu_{i+1j-1k} + \mu_{i-1j+1k} + && (48)\\
\mu_{i+1jk+1} + \mu_{i-1jk-1} + \mu_{i+1jk-1} + \mu_{i-1jk+1} + \mu_{ij+1k+1} + \mu_{ij-1k-1} + &\mu_{ij+1k-1} + \mu_{ij-1k+1}\Big\}. && (49)
\end{aligned}$$

This can be combined to give an easily recognisable finite difference solution for the diffusion equation

$$\begin{aligned}
\phi_{ijk}^{n+1} = \phi_{ijk}^n + 2Mw_2/2c_s^4\Big\{&\mu_{i+1j+1k} - 2\mu_{ijk} + \mu_{i-1j-1k} + \mu_{i+1j-1k} - 2\mu_{ijk} + \mu_{i-1j+1k} + && (50)\\
&\mu_{i+1jk+1} - 2\mu_{ijk} + \mu_{i-1jk-1} + \mu_{i+1jk-1} - 2\mu_{ijk} + \mu_{i-1jk+1} + && (51)\\
&\mu_{ij+1k+1} - 2\mu_{ijk} + \mu_{ij-1k-1} + \mu_{ij+1k-1} - 2\mu_{ijk} + \mu_{ij-1k+1}\Big\}. && (52)
\end{aligned}$$

Note that this uses the twelve stencil points in μ related to the velocities with weight w_2 and so should have very good isotropy properties. The final prefactor is M (check).

7.8 User input

```
free_energy symmetric
```

The following applies for the binary fluid problem with compositional order parameter ϕ and free energy (excluding the term in the density ρ):

$$F[\phi] = \int dr \left(\tfrac{1}{2}A\phi^2 + \tfrac{1}{4}B\phi^4 + \tfrac{1}{2}\kappa(\nabla\phi)^2\right). \tag{53}$$

This is described in some detail by Kendon et al [?]. The first two terms represent the bulk contribution, whereas the term in κ penalises curvature in the interface.

$\boxed{\texttt{A}}$ The parameter A. Note $A < 0$.

$\boxed{\texttt{B}}$ The parameter B. Note $B = -A$ for common usage, although this is not enforced.

$\boxed{\texttt{K}}$ The parameter κ, which is positive.

The order parameter evolution is determined by a Cahn-Hilliard equation with mobility set by

$\boxed{\texttt{mobility}}$ Sets mobility M (uniform in space).

The following parameters control the initialisation of the order parameter.

$\boxed{\texttt{phi\_initialisation}}$

Determines how the compositional order parameter is initialised at the start of the run. If set to `spinodal` the value is set to ϕ_0 as set above plus or minus a random noise, the magnitude of which is set by the value of the `noise` key. If set to `block` a one-dimensional profile is set up in the z-direction representing two blocks of fluid with $\phi = \pm 1$. The two interfaces are set at $z = L_z/4$ and $z = 3L_z/4$ with the equilibrium $\tanh(z/\xi_0)$ profile having appropriate width. The $\phi = -1$ section is in the middle.

$\boxed{\texttt{phi0}}$

The mean compositional order parameter roughly $-0.5 < \phi_0 < 0.5$. The default value is zero, i.e., a symmetric 50:50 mixture by volume.

$\boxed{\texttt{noise}}$

The magnitude of the initial fluctuations in ϕ used to initiate spinodal decomposition.

7.8.1 Binary fluid using two distributions

$\boxed{\texttt{symmetric\_lb}}$

This is the special case where the composition is represented by a second LB distribution, and an appropriate lattice kinetic equation approximating the Cahn-Hilliard equation is solved. In this case the above parameters have the same meaning.

Note that the relaxation time for the second distribution is related to the mobility by $tau_\phi = M\rho_0/\Delta t + 1/2$.

7.8.2 Brazovskii

$\boxed{\texttt{free\_energy brazovskii}}$

This is similar to the symmetric free energy, but with one extra term in a higher derivative of ϕ.

$$F[\phi] = \int dr \left(\tfrac{1}{2}A\phi^2 + \tfrac{1}{4}B\phi^4 + \tfrac{1}{2}\kappa(\nabla\phi)^2 + \tfrac{1}{2}C(\nabla^2\phi)^2 \right). \tag{54}$$

The parameters now include C. For $A < 0$, phase separation occurs with a result depending on κ: ones get two symmetric phases for $\kappa > 0$ (cf. symmetric) or a lamellar phase for $\kappa < 0$.

$\boxed{\texttt{A}}$ Bulk parameter $A < 0$.

$\boxed{\texttt{B}}$ Bulk parameter $B = -A$.

$\boxed{\texttt{K}}$ Negative for lamellar phase.

$\boxed{\texttt{C}}$ Positive.

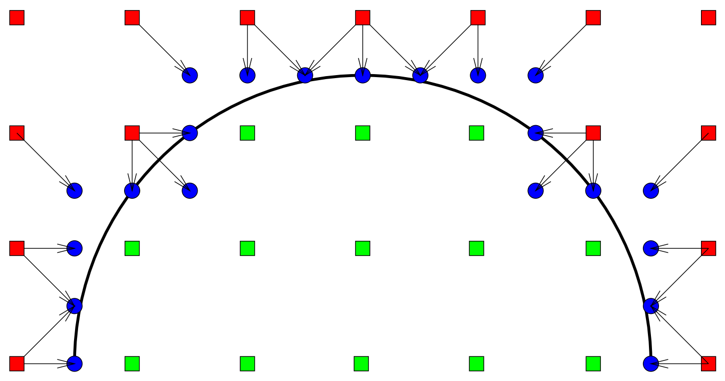


Figure 1: Boundary links for a colloidal particle in two dimensions with D2Q9 (half the particle is shown). Links join fluid sites (red) to solid sites (green) and intersect the circular shell radius a_0. Boundary nodes (circles) lie exactly half way between pairs of lattice nodes. The generalisation to three dimensions is straightforward.

Other parameters, including the mobility, are set as for the symmetric free energy (see above).

8 Colloids

8.1 General

Colloidal particles are assumed to be spherical with a geometrical centre $\mathbf{r}_c$, which is also the centre of mass. The centre is allowed to move continuously across the lattice with velocity $\mathbf{U}$; the particle has an angular velocity $\mathbf{\Omega}$. The surface of the colloid is defined by an input radius, a_0, which determines which lattice nodes are inside or outside the colloid. (The hydrodynamic properties of the colloid are specified by a different radius a_h — more of this later.) The boundary links are then the set of vectors joining lattice nodes which intersect the spherical surface $\{\mathbf{c}_b\}$. A schematic picture is presented in Figure 1. Note that a lattice node exactly at the solid-fluid interface is defined to be outside the colloid.

In the original approach of Ladd, fluid occupied nodes both inside and outside the particle. The effect of the "internal fluid" is known to be restricted to short time scales (compared to the characteristic time a_0^2/ν), on which the fluid inside the particle relaxes to a solid body rotation [19]. There are a number of problems related to the internal fluid, so we use fully solid particles.

Nguyen and Ladd.

For each boundary link, it is useful to define a boundary vector, $\mathbf{r}_b$, which connects the centre of the colloid to the appropriate boundary node (note that the length of this vector is not necessarily equal to the input radius of the colloid).

A *boundary node* halfway between fluid nodes which are joined by a boundary link. The position of the boundary node is always exactly halfway along the boundary link $(\mathbf{r} + \frac{1}{2}\mathbf{c}_b\Delta t)$ regardless of the actual position of the intersection of the colloid surface and the boundary link. The colloid therefore has a discrete representation which becomes a better approximation to the sphere as the input radius becomes larger; this approximation is known to be reasonably good for $a_0 > 5\Delta x$.

8.2 Bounce-back on links

The standard boundary condition required for the solid-fluid interface of a moving particle is described by Ladd [25], and is generally refered to as bounce-back on links (BBL). A boundary link is defined as joining a node $\mathbf{r}$ inside the particle to one outside at $\mathbf{r}+\mathbf{c}_b\Delta t$. If the post-collision distributions are denoted by f^*, then the distributions must be reflected at the solid surface so that

$$f_{b'}(\mathbf{r};t+\Delta t) = f_b^*(\mathbf{r};t) - \frac{2w_{c_b}\rho\mathbf{u}_b.\mathbf{c}_b}{c_s^2} \tag{55}$$

where the boundary link $\mathbf{c}_{b'} = -\mathbf{c}_b$. The velocity at the boundary

$$\mathbf{u}_b = \mathbf{U} + \mathbf{\Omega}\times\mathbf{r}_b \tag{56}$$

is determined by the particle linear velocity $\mathbf{U}$ and angular velocity $\mathbf{\Omega}$. The change in momentum described by

The force exerted on a single link is

$$\mathbf{F}_b(\mathbf{r}+\tfrac{1}{2}\mathbf{c}_b\Delta t; t+\tfrac{1}{2}\Delta t) = \frac{\Delta x^3}{\Delta t}\Big[2f_b^*(\mathbf{r};t) - \frac{2w_{c_b}\rho_0\mathbf{u}_b.\mathbf{c}_b}{c_s^2}\Big]\mathbf{c}_b, \tag{57}$$

with corresponding torque $\mathbf{T}_b = \mathbf{r}_b\times\mathbf{F}_b$.

The total hydrodynamic force on the particle is then found by taking the sum of $\mathbf{F}_b$ over all the boundary links defining the particle. The is an associated torque on each link of $\mathbf{r}_b\times\mathbf{F}_b$, which again is summed over all links to give the total torque on the colloid.

Note that ρ in the above equations is the density of the fluid at the appropriate fluid node for the link. As the density fluctuations in the fluid are small compared with the mean density ρ_0, the density in the correction to the bounce-back can be replaced by the mean ρ_0.

8.3 Dynamics

Having computed the total force and torque on an individual colloid, is is possible to update the the linear velocities

$$m_0\mathbf{U}(t+\Delta t) = m_0\mathbf{U}(t) + \Delta t\mathbf{F}(t), \tag{58}$$

where the mass of the colloid is related to the input radius by $m_0 = \frac{4}{3}\pi\rho_0 a_0^3$ and the angular velocity

$$I_0\mathbf{\Omega}(t+\Delta t) = I_0\mathbf{\Omega}(t) + \Delta t\mathbf{T}(t), \tag{59}$$

where the moment of inertia is $I_0 = \frac{2}{5}m_o a_o^2$. However, this explicit update is generally found to have poor stability properties [25, **?**]. The alternative is to use a velocity update which is implicit [19, **?**].

The total force and torque on a colloid can be split into velocity-dependent and -independent parts by combining Equations (56) and (57) to eliminate the boundary velocity $\mathbf{u}_b$. In this way, the decomposition is

$$\mathbf{F} = \mathbf{F}_0 - \boldsymbol{\zeta}^{FU}.\mathbf{U} - \boldsymbol{\zeta}^{F\Omega}.\mathbf{\Omega}, \tag{60}$$

$$\mathbf{T} = \mathbf{T}_0 - \boldsymbol{\zeta}^{TU}.\mathbf{U} - \boldsymbol{\zeta}^{T\Omega}.\mathbf{\Omega}. \tag{61}$$

The velocity independent parts of the force and the torque (appropriate for a colloid at rest) are

$$\mathbf{F}_0(t+\tfrac{1}{2}\Delta t) = \frac{2\Delta x^3}{\Delta t}\sum_b \big[f_b^*(\mathbf{r};t) - f_{b'}^*(\mathbf{r}+\mathbf{c}_i\Delta t;t)\big]\mathbf{c}_b, \tag{62}$$

$$\mathbf{T}_0(t+\tfrac{1}{2}\Delta t) = \frac{2\Delta x^3}{\Delta t}\sum_b \big[f_b^*(\mathbf{r};t) - f_{b'}^*(\mathbf{r}+\mathbf{c}_i\Delta t;t)\big](\mathbf{r}_b\times\mathbf{c}_b) \tag{63}$$

The matrices $\boldsymbol{\zeta}$ are interpreted as drag coefficients and can be written as

$$\boldsymbol{\zeta}^{FU} = \frac{4\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}\mathbf{c}_b\mathbf{c}_b, \tag{64}$$

$$\boldsymbol{\zeta}^{F\Omega} = \frac{4\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}\mathbf{c}_b(\mathbf{r}_b\times\mathbf{c}_b), \tag{65}$$

$$\boldsymbol{\zeta}^{TU} = \frac{4\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}(\mathbf{r}_b\times\mathbf{c}_b)\mathbf{c}_b, \tag{66}$$

$$\boldsymbol{\zeta}^{T\Omega} = \frac{4\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}(\mathbf{r}_b\times\mathbf{c}_b)(\mathbf{r}_b\times\mathbf{c}_b). \tag{67}$$

As an example, the full form of the $\boldsymbol{\zeta}^{F\Omega}$ matrix is

$$\begin{pmatrix} \sum_b c_{bx}(\mathbf{r}_b\times\mathbf{c}_b)_x & \sum_b c_{bx}(\mathbf{r}_b\times\mathbf{c}_b)_y & \sum_b c_{bx}(\mathbf{r}_b\times\mathbf{c}_b)_z \\ \sum_b c_{by}(\mathbf{r}_b\times\mathbf{c}_b)_x & \sum_b c_{by}(\mathbf{r}_b\times\mathbf{c}_b)_y & \sum_b c_{by}(\mathbf{r}_b\times\mathbf{c}_b)_z \\ \sum_b c_{bz}(\mathbf{r}_b\times\mathbf{c}_b)_x & \sum_b c_{bz}(\mathbf{r}_b\times\mathbf{c}_b)_y & \sum_b c_{bz}(\mathbf{r}_b\times\mathbf{c}_b)_z \end{pmatrix}. \tag{68}$$

If the particle has a symmetric distribution of boundary links (the special case where the centre of mass is on a symmetry point of the lattice) the $\boldsymbol{\zeta}^{FU}$ and $\boldsymbol{\zeta}^{T\Omega}$ matrices are diagonal, while the $\boldsymbol{\zeta}^{F\Omega}$ and $\boldsymbol{\zeta}^{TU}$ matrices are zero.

The velocity updates can now be rewritten using an implicit update, which results in

$$m_0\mathbf{U}(t+\Delta t) = m_0\mathbf{U}(t) + \Delta t\big[\mathbf{F}_0(t+\tfrac{1}{2}\Delta t) - \boldsymbol{\zeta}^{FU}.\mathbf{U}(t+\Delta t) - \boldsymbol{\zeta}^{F\Omega}.\boldsymbol{\Omega}(t+\Delta t)\big], \tag{69}$$

$$I_0\boldsymbol{\Omega}(t+\Delta t) = I_0\boldsymbol{\Omega}(t) + \Delta t\big[\mathbf{T}_0(t+\tfrac{1}{2}\Delta t) - \boldsymbol{\zeta}^{TU}.\mathbf{U}(t+\Delta t) - \boldsymbol{\zeta}^{T\Omega}.\boldsymbol{\Omega}(t+\Delta t)\big]. \tag{70}$$

In general all the matrix elements are non-zero, in which case equations (69) and (70) can be solved via a 6×6 matrix inversion.

Finally, the position of the particle can be updated: an Euler forward step is taken using the updated velocity

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \Delta t\mathbf{U}(t+\Delta t) \tag{71}$$

8.4 Changes in particle shape

One consequence is that the discrete shape of the particle fluctuates as the colloid centre moves relative to the lattice. This means that the size of the particle as seen by the fluid changes despite the fact that the input radius of the colloid is fixed. However, these fluctuations are small for input radii greater than about 5 lattice units [26]. Furthermore, the fluctuations may be greater for some input radii than others [**?**].

In the standard approach, changes in the map of boundary links are accomodated by the internal fluid. If the particle shape changes so that an internal node is exposed, the internal fluid at that site rejoins the fluid proper with the solid body momentum (assuming the internal fluid is relaxed to solid-body rotation). If a fluid node is covered by particle movement, then it simply joins the internal fluid, and will relax to solid-body rotation. Without internal fluid, changes in particle shape in which fluid nodes are either covered or uncovered must be accompanied by the removal or addition of fluid with appropriate properties at the nodes in question. It is essential that this is done in a way which minimises the perturbation to the fluid flow.

If a fluid node at $\mathbf{r}$ is covered by the movement of a solid particle, the fluid loses a mass $\rho(\mathbf{r})\Delta x^3$, of which an excess $\Delta M_c = (\rho(\mathbf{r}) - \rho_0)\Delta x^3$ must be replaced explicitly so that the overall mass density is unchanged. At the same time, the colloid must assume the momentum lost by the fluid $\Delta x^3 \sum_i f_i(\mathbf{r})\mathbf{c}_i$.

Similarly, when a fluid node is exposed by particle movement, fluid must be replaced with the appropriate distribution, density and velocity. If the new density is ρ (to be determined by some method) then the fluid gains an excess of mass $\Delta M_u = (\rho - \rho_0)\Delta x^3$ which must be balanced elsewhere. If the new fluid is assumed to have velocity $\mathbf{u}$ then the particle must give up momentum $\Delta x^3 \rho \mathbf{u}$.

Following Nguyen and Ladd [?], these changes owing to change in particle shape are implemented by added a small correction to the bounce-back at the following time step. The excess mass from a covered fluid site is redistributed over all the boundary nodes by redefined the mometum transfer at bounce-back

$$f_{b'}(\mathbf{r}; t + \Delta t) = f_b^*(\mathbf{r}; t) - \frac{2w_{c_b}\rho_0 \mathbf{u}_b.\mathbf{c}_b}{c_s^2} + \frac{w_{c_b}\rho_0 \Delta M_c}{A}, \tag{72}$$

where $A = \rho_0 \Delta x^3 \sum_b w_{c_b}$. The accompanying force on the particle owing to the change in shape is then that lost by the fluid plus the contributions from the final term in (72) summed over all the boundary links

$$\Delta \mathbf{F}_c = \frac{\Delta x^3}{\Delta t}\sum_i f_i \mathbf{c}_i + \frac{\Delta x^3}{\Delta t}\sum_b \frac{w_{c_b}\rho_0 \Delta M_c}{A}\mathbf{c}_b. \tag{73}$$

Note that for a closed surface, $\sum_b w_{c_b}\mathbf{c}_b = 0$, and the second term on the right-hand side of Eq. (73) is zero, i.e., the redistribution of mass does not contribute to the net force and torque on the particle. The corresponding change in torque is

$$\Delta \mathbf{T}_c = \frac{\Delta x^3}{\Delta t}\mathbf{r}_c \times \sum_i f_i \mathbf{c}_i + \frac{\Delta x^3}{\Delta t}\sum_b \frac{w_{c_b}\rho_0 \Delta M_c}{A}\mathbf{r}_b \times \mathbf{c}_b, \tag{74}$$

where $\mathbf{r}_c$ is the boundary vector at the position where the fluid has been removed. Again, for a closed surface, the redistribution of mass does not contribute to the net torque on the particle.

Likewise, the contribution to the bounce-back from newly uncovered nodes is

$$-\frac{w_{c_b}\rho_0 \Delta M_u}{A} \tag{75}$$

leading to a change in force of

$$\Delta \mathbf{F}_u = -\frac{\Delta x^3 \rho(\mathbf{r})\mathbf{u}(\mathbf{r})}{\Delta t} - \frac{\Delta x^3}{\Delta t}\sum_b \frac{w_{c_b}\rho_0 \Delta M_u}{A}\mathbf{c}_b, \tag{76}$$

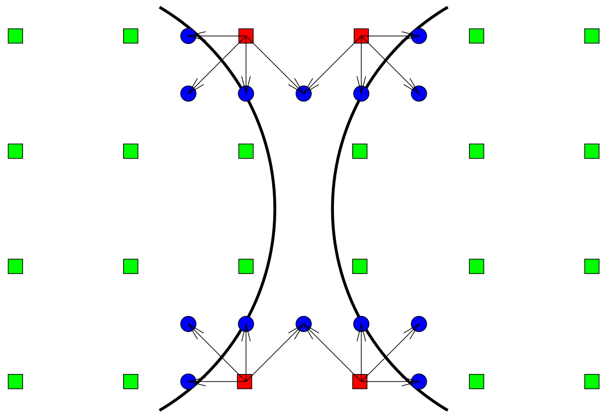


Figure 2: Two colloids close to contact are missing links in the region of closest approach owing to a lack of fluid nodes in the interstice.

with a corresponding change in the torque. If more than one lattice node is either covered or uncovered by the movement of the particle, then these contributions add in a simple way. The overall contributions can be added to the velocity-independent terms $\mathbf{F}_0$ and $\mathbf{T}_0$ appearing in Equations (60) and (61).

8.5 Particles near contact

Particles near contact may not pocess a full set of boundary links (see Figure 2). This leads to a potential non-conservation of mass associated with the particle motion which must be corrected.

Again following Nguyen and Ladd [**?**], mass conservation is enforced explicitly using the following procedure. There is a mass transfer associated with the bounce-back at each link of $(2w_{c_b}\rho_0\mathbf{u}_b.\mathbf{c}_b/c_s^2)\Delta x^3$. The net mass transport into the particle is the sum of this quantity over all the links, and can be written

$$\Delta M_s = -\frac{2\Delta x^3\rho_0}{c_s^2}\Big[\mathbf{U}.\sum_b w_{c_b}\mathbf{c}_b + \mathbf{\Omega}.\sum_b w_{c_b}\mathbf{r}_b\times\mathbf{c}_b\Big]. \tag{77}$$

For any partilce with a full set of boundary links, both $\sum_b w_{c_b}\mathbf{c}_b$ and $\sum_b w_{c_b}\mathbf{r}_b\times\mathbf{c}_b$ are zero. However, if the set of boundary links is imcomplete (as in Figure 2) ΔM_s is not zero and a correction is required. This is again made by redistributing the excess or deficit of mass over the other existing boundary nodes. The additional contribution to the bounce-back here is $-w_{c_b}\rho_0\Delta M_s/A$, where $A = \rho_0\Delta x^3\sum_b w_{c_b}$ (cf. Equation 72).

This contribution now adds to the velocity-dependent part of the force and torque and leads to a slightly different form of the drag matrices

$$\boldsymbol{\zeta}^{FU} = \frac{-2\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}(\mathbf{c}_b - \overline{\mathbf{c}_b})\mathbf{c}_b \tag{78}$$

$$\boldsymbol{\zeta}^{F\Omega} = \frac{-2\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}\mathbf{c}_b(\mathbf{r}_b\times\mathbf{c}_b - \overline{\mathbf{r}_b\times\mathbf{c}_b}), \tag{79}$$

$$\boldsymbol{\zeta}^{TU} = \frac{-2\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}(\mathbf{r}_b\times\mathbf{c}_b)(\mathbf{c}_b - \overline{\mathbf{c}_b}), \tag{80}$$

$$\boldsymbol{\zeta}^{T\Omega} = \frac{-2\rho_0\Delta x^3}{c_s^2\Delta t}\sum_b w_{c_b}(\mathbf{r}_b\times\mathbf{c}_b)(\mathbf{r}_b\times\mathbf{c}_b - \overline{\mathbf{r}_b\times\mathbf{c}_b}). \tag{81}$$

The mean quantities

$$\overline{\mathbf{c}_b} = \frac{\sum_b w_{c_b}\mathbf{c}_b}{\sum_b w_{c_b}} \tag{82}$$

and

$$\overline{\mathbf{r}_b\times\mathbf{c}_b} = \frac{\sum_b w_{c_b}\mathbf{r}_b\times\mathbf{c}_b}{\sum_b w_{c_b}}. \tag{83}$$

A little algebra will show that both occurances of $\mathbf{c}_b$ and/or $\mathbf{r}_b\times\mathbf{c}_b$ in the definition of the drag matrices can be replaced by their deviation from the mean. This provides a convenient way to compute the drag matrices (maintaining symmetry) and computing the correct force and torque on the particle when close to contact.

8.6 Rotational motions

For a number of applications it is useful to update not only the psoition of a spherical particle, but its orientation. This is important, e.g., when considering magnetic particles. The best way to implementment the required rotations is by considering quaternions.

A quaternion can be thought of as an extended complex number

$$q = w + xi + yj + zk \tag{84}$$

where $i^2 = j^2 = k^2 = -1$ and $ijk = -1$. The norm, or length of a quaternion is

$$N(q) = (w^2 + x^2 + y^2 + z^2)^{1/2}. \tag{85}$$

Addition and subtraction of quaternions proceeds as normal, but care is required for multiplication, which is associative but not commutative. If the quaternion is writen as $q = (w, \mathbf{v})$ where $\mathbf{v}$ is a normal 3-vector, then multiplication can be written

$$q_1q_2 = (w_1w_2 - \mathbf{v}_1.\mathbf{v}_2, w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1\times\mathbf{v}_2), \tag{86}$$

where the usual scalar and vector products are used. Note that the conjugate of a quaterion $q^\star = (w, -\mathbf{v})$ so that $qq^\star = N^2(q)$. It can be shown that a rotation of an arbitrary vector though an angle θ around the unit axis of rotaion $\hat{\mathbf{w}}$ can be represented by the quarternion

$$q = (\cos(\theta/2), \hat{\mathbf{w}}\sin(\theta/2)) \tag{87}$$

which ensures that $N(q) = 1$. If the arbitrary vector is $(0, \mathbf{v})$, then the rotated vector $\mathbf{v}^{'}$ can be written as

$$(0, \mathbf{v}^{'}) = q(0, \mathbf{v})q^\star \tag{88}$$

which can be expanded as

$$\mathbf{v}^{'} = (1-\cos\theta)(\mathbf{v}.\hat{\mathbf{w}})\hat{\mathbf{w}} + \mathbf{v}\cos\theta + (\hat{\mathbf{w}}\times\mathbf{v})\sin\theta. \tag{89}$$

8.7 Colloid-Colloid interactions

8.7.1 Lubrication corrections

8.7.2 Hard sphere interaction

Particles do not overlap, i.e., there is always a hard-sphere interaction which is a function of the separation h:

$$v^{hs}(h) = \begin{cases} \infty & h \leq 0, \\ 0 & h > 0. \end{cases} \tag{90}$$

The hard-sphere interaction does not give rise to a force.

8.7.3 Soft-sphere interaction

A simple soft-sphere interaction is available, the basic form of which is:

$$v^{ss}(r) = \epsilon(\sigma/r)^{\nu}, \tag{91}$$

where ϵ sets the energy scale, and σ sets the characteristic width. The steepness of the potential is set by the exponent $\nu(> 0)$.

To prevent the need for calculation of long-range interactions, the soft-sphere potential is truncated in a "cut-and-shift" approach. This is done in such a way as to smoothly match both the potential and the force at the cut-off distance r_c. For $r < r_c$ the potential is then

$$v^{ss}(r) - v^{ss}(r_c) - (r - r_c) \left. \frac{dv^{ss}}{dr} \right|_{r=r_c}. \tag{92}$$

Clearly, this potential is not exactly what we first thought of as soft-sphere. Matching the potential smoothly ensures conservation of energy, while matching the force smoothly prevents potential instabilities in the molecular dynamics update.

The soft-sphere potential can be useful as a mechanism to keep the particles from touching (which risks hard-sphere interactions), in which case is is possible to compute the interaction as a function of the separation h, instead of r. The force between two particles is computed via the derivative of equation 92 with respect to r:

$$\mathbf{F}_{ij}(r) = - \left\{ \frac{dv^{ss}(r)}{dr} - \left. \frac{dv^{ss}(r)}{dr} \right|_{r=r_c} \right\} \hat{\mathbf{r}}_{ij} \tag{93}$$

where $\hat{\mathbf{r}}_{ij}$ is the unit vector joining the centre of particle i to the centre of j.

8.7.4 Leonard-Jones interaction

8.8 Cholesteric Anchoring

We have a fluid free energy density

$$f = \tfrac{1}{2}\kappa_0(\partial_\beta Q_{\alpha\beta})^2 + \tfrac{1}{2}\kappa_1(\epsilon_{\alpha\gamma\sigma}\partial_\gamma Q_{\sigma\beta} + 2q_0 Q_{\alpha\beta})^2 \tag{94}$$

where we have ignored the bulk terms for the time being, but we retain two elastic constants κ_0 and κ_1 in the distortion term. The cholesteric pitch $p = 2\pi/q_0$.

There is also a surface term (an area density)

$$f_s = \tfrac{1}{2}w_1(Q_{\alpha\beta} - Q^0_{\alpha\beta})^2 \tag{95}$$

where $Q^0_{\alpha\beta}$ is the preferred order parameter configuration at the surface (in the case of normal anchoring or fixed planar anchoring). For degenerate planar anchoring, following Fournier and Galatola [14], we have

$$f_s = \tfrac{1}{2}w_1(\tilde{Q}_{\alpha\beta} - \tilde{Q}^{\perp}_{\alpha\beta})^2 + \tfrac{1}{2}w_2(\tilde{Q}_{\alpha\beta}\tilde{Q}_{\alpha\beta} - S_0^2)^2 \tag{96}$$

where $\tilde{Q}_{\alpha\beta}{+}(1/3)S_0\delta_{\alpha\beta}$, S_0 is is a fixed surface amplitude; to obtain a prefered degenerate configuration we take the fluid order parameter $Q_{\alpha\beta}$ and project $\tilde{Q}^{\perp}_{\alpha\beta} = P_{\alpha\gamma}\tilde{Q}_{\gamma\sigma}P_{\sigma\beta}$, where $P_{\alpha\beta} = \delta_{\alpha\beta} - n_\alpha n_\beta$ and n_α is the local unit surface normal.

The boundary condition we wish to apply are the Euler-Lagrange equations. They are at the surface of the particle

$$n_\gamma \frac{\partial f}{\partial Q_{\alpha\beta,\gamma}} = 0 \tag{97}$$

where $Q_{\alpha\beta,\gamma} = \partial_\gamma Q_{\alpha\beta}$. Here, n_γ is the outward unit normal at the surface (pointing into the fluid) [42]. Note that compared to the regular minimization as explained in [43] this is a surface integral term that would usually vanish in the bulk. With the addition of the surface term we obtain

$$n_\gamma \frac{\partial f}{\partial Q_{\alpha\beta,\gamma}} + \frac{\partial f_s}{\partial Q_{\alpha\beta}} = 0. \tag{98}$$

The derivative of the distortion free energy with respect to $Q_{\alpha\beta,\gamma}$ gives

$$\kappa_0 n_\beta \partial_\gamma Q_{\alpha\gamma} + \kappa_1 n_\gamma(\partial_\gamma Q_{\alpha\beta} - \partial_\alpha Q_{\gamma\beta}) - 2\kappa_1 q_0 n_\gamma \epsilon_{\alpha\gamma\sigma} Q_{\sigma\beta}. \tag{99}$$

However, if we want a symmetric form (the derivative with respect to $Q_{\beta\alpha,\gamma}$ is just as good) we can write this as

$$\begin{aligned}\tfrac{1}{2}\kappa_0(n_\alpha\partial_\gamma Q_{\beta\gamma} + n_\beta\partial_\gamma Q_{\alpha\gamma}) + \kappa_1 n_\gamma\partial_\gamma Q_{\alpha\beta} - \tfrac{1}{2}\kappa_1 n_\gamma(\partial_\alpha Q_{\gamma\beta} + \partial_\beta Q_{\gamma\alpha})\\ -\kappa_1 q_0 n_\gamma(\epsilon_{\alpha\gamma\sigma}Q_{\sigma\beta} + \epsilon_{\beta\gamma\sigma}Q_{\sigma\alpha}).\end{aligned} \tag{100}$$

If we add the derivative of the surface free energy, where we assume that the preferred orientation $Q^0_{\alpha\beta}$ is independent of $Q_{\alpha\beta}$, we get a full boundary condition for the gradient of the tensor order parameter at the solid-fluid boundary:

$$\begin{aligned}\tfrac{1}{2}\kappa_0(n_\alpha\partial_\gamma Q_{\beta\gamma} + n_\beta\partial_\gamma Q_{\alpha\gamma}) + \kappa_1 n_\gamma\partial_\gamma Q_{\alpha\beta} - \tfrac{1}{2}\kappa_1 n_\gamma(\partial_\alpha Q_{\gamma\beta} + \partial_\beta Q_{\gamma\alpha})\\ -\kappa_1 q_0 n_\gamma(\epsilon_{\alpha\gamma\sigma}Q_{\sigma\beta} + \epsilon_{\beta\gamma\sigma}Q_{\sigma\alpha}) - w_1(Q_{\alpha\beta} - Q^0_{\alpha\beta}) = 0.\end{aligned} \tag{101}$$

Note that for planar generate anchoring, the final term (the 'surface molecular field') is replaced by

$$-w_1(\tilde{Q}_{\alpha\beta} - \tilde{Q}^{\perp}_{\alpha\beta}) - 2w_2(\tilde{Q}_{\gamma\sigma}\tilde{Q}_{\gamma\sigma} - S_0^2)\tilde{Q}_{\alpha\beta}. \tag{102}$$

8.8.1 Discrete implementation

In three dimensions, the boundary condition Eq. 101 provides six equations containing (potentially) 18 unknown derivatives $\partial_\gamma Q_{\alpha\beta}$ corresponding to the 6 independent elements of the order parameter tensor Q_{xx}, Q_{xy}, Q_{xz}, Q_{yy}, Q_{yz}, Q_{zz}. Note that the equation corresponding to Q_{zz} must appear to retain isotropy. However, Q_{zz} itself, and its derivatives, may be replaced via the constraint on the trace of $Q_{\alpha\beta}$. We can therefore either solve a fully determined system including Q_{zz}, and then impose tracelessness on the result, or replace Q_{zz} and solve six equations for five unknowns, with the sixth equation acting as the constraint. These methods provide the same answer for cases where the surface normal is along the coordinate directions.

At a flat surface with, e.g., $n = (1, 0, 0)$, the number of unknowns are the gradients $\partial_x Q_{\alpha\beta}$ at the boundary if we assume the tangential gradients $\partial_y Q_{\alpha\beta}$ and $\partial_z Q_{\alpha\beta}$ can be approximated using the standard differencing method involving only fluid values of $Q_{\alpha\beta}$. We proceed by computing the constant terms

$$-\kappa_1 q_0 n_\gamma (\epsilon_{\alpha\gamma\sigma} Q_{\sigma\beta} + \epsilon_{\beta\gamma\sigma} Q_{\sigma\alpha}) - w(Q_{\alpha\beta} - Q^0_{\alpha\beta})$$

using $Q_{\alpha\beta}$ from the adjacent fluid site, and an estimate of $Q^0_{\alpha\beta}$ for the appropriate anchoring type. To these constant terms are added the tangential gradients. The gradients at the surface are then computed by solving a 5x5 linear algebra problem for $\partial_x Q_{\alpha\beta}$. This allows the full gradient at the adjacent fluid site to be constructed.

At concave edges or corners, where it is not possible to compute the tangential gradients from the usual stencil as for a flat interface, a different approach is required. We note than an attempt to solve a 10x10 or 15x15 linear algebra problem for the full set of unknown gradients has proven unreliable: in the case where n_γ is normal to the coordinate directions there is simply no solution available. Instead we adopt an iterative approach.

Consider the case of an edge where the solid neighbours are in the x and y directions. We first make an approximation to the tangential gradients in the x and y directions by using a one-sided derivative from the fluid points available. This tangential approximation in y is then used to find a first estimate of the gradients $\partial_x Q_{\alpha\beta}$ using the procedure described above. The tangential estimate in x is then used to obtain $\partial_y Q_{\alpha\beta}$. These surface gradients are then used to improve the estimate of the tangential gradients, and the process is iterated.

8.9 User input

The are a number of important options required for colloids. If none of these keys are present, no colloids will be used.

```
colloid_init
```

This determines how the code initialises particles (or not). As stated, the default is to have no particles (value `no_colloids`). There are two other options at the moment.

There are two options:

```
colloid_init from_file
```

requires a preprepared file of colloid information. To understand how to create such a file in the correct format, see the example program `colloid_file.c` in the util directory.

```
colloid_init random
```

This may be useful to initialise a limited number of colloids at low volume fraction. Note that there is no guarantee that the particles do not overlap, in which case the initialisation will terminate. The number of particles and their properties are determined by the value of the keys described below.

```
colloid_type
```

This controls the the type of particle. Appropriate values are `inactive`, giving a standard fully resolved LB colloid; `active` switches on the correction to BBL appropriate for active particles (expecting parameters b_1 and b_2 to be set); `subgrid` gives unresolved particles (radius $<$ 1 lattice unit) and no BBL.

	$Q_{xx,x}$	$Q_{xy,x}$	$Q_{xz,x}$	$Q_{yy,x}$	$Q_{yz,x}$	$Q_{zz,x}$
Q_{xx}	$\kappa_0 n_x$	$-\kappa_1 n_y$	$-\kappa_1 n_z$			
Q_{xy}	$\kappa_0 n_y$	$\kappa' n_x$		$-\kappa_1 n_y$	$-\kappa_1 n_z$	
Q_{xz}	$\kappa_0 n_z$		$\kappa' n_x$		$-\kappa_1 n_y$	$-\kappa_1 n_z$
Q_{yy}		$\kappa_0 n_y$		$\kappa_1 n_x$		
Q_{yz}		$\kappa_0 n_z$	$\kappa_0 n_y$		$2\kappa_1 n_x$	
Q_{zz}			$\kappa_0 n_z$			$\kappa_1 n_x$

	$Q_{xx,y}$	$Q_{xy,y}$	$Q_{xz,y}$	$Q_{yy,y}$	$Q_{yz,y}$	$Q_{zz,y}$
Q_{xx}	$\kappa_1 n_y$	$\kappa_0 n_x$				
Q_{xy}	$-\kappa_1 n_x$	$\kappa' n_y$	$-\kappa_1 n_z$	$\kappa_0 n_x$		
Q_{xz}		$\kappa_0 n_z$	$2\kappa_1 n_y$		$\kappa_0 n_x$	
Q_{yy}		$-\kappa_1 n_x$		$\kappa_0 n_y$	$-\kappa_1 n_z$	
Q_{yz}			$-\kappa_1 n_x$	$\kappa_0 n_z$	$\kappa' n_y$	$-\kappa_1 n_z$
Q_{zz}					$\kappa_0 n_z$	$\kappa_1 n_y$

	$Q_{xx,z}$	$Q_{xy,z}$	$Q_{xz,z}$	$Q_{yy,z}$	$Q_{yz,z}$	$Q_{zz,z}$
Q_{xx}	$\kappa_1 n_z$		$\kappa_0 n_x$			
Q_{xy}		$2\kappa_1 n_z$	$\kappa_0 n_y$		$\kappa_0 n_x$	
Q_{xz}	$-\kappa_1 n_x$	$-\kappa_1 n_y$	$\kappa' n_z$			$\kappa_0 n_x$
Q_{yy}				$\kappa_1 n_z$	$\kappa_0 n_y$	
Q_{yz}		$-\kappa_1 n_x$		$-\kappa_1 n_y$	$\kappa' n_z$	$\kappa_0 n_y$
Q_{zz}			$-\kappa_1 n_x$		$-\kappa_1 n_y$	$\kappa_0 n_z$

Table 4: Coefficients of the various derivatives of the order parameter tensor appearing in six equations for the elements of the order parameter (including Q_{zz}). Note $\kappa_0 + \kappa_1 = \kappa'$ and all the coefficients have been multiplied by a factor of 2 in the off-diagonal equations.

If more than one particle is used, all positions are set at random.

```
colloid_gravity 0.0_0.0_-0.0001
```

Sets a body force on each particle. Useful, for example, for sedimentation. For example, the above gives a constant force in the negative z-direction on each particle. The code automatically computes the compensating body force on fluid nodes required to give no net change in the total momentum of the system at each time step.

8.9.1 Interactions and the cell list

In each local domain, colloidal particles are stored in a cell list. This is a common structure in molecular dynamics-like problems. Here, it also serves as the basis for parallel communication of colloid information, so there are a number of constraints which it is important to understand. In particular, parallel communication imposes the constraint that there be at least 2 cells in the local domain in each coordinate direction.

The size of the cells is computed automatically by the code based on the local domain size, particle size, and the cut-off distance of any pairwise interactions that are relevant. This will usually mean trying to maximise the number of cells (to reduce pairwise interaction calculation). The user must set the minimum allowable cell width.

```
colloid_cell_min
```

This sets the minimum cell list width. This key must be set and there must be at least two cells per processor domain. If the minimum does not capture the specified interactions, a fatal error will result. For spherical particles, the minimum width will typically be $2a_h + h_c$ where h_c is an interaction cut-off distance.

In the case where very large particles are required, it can be useful to switch of the cell list. This is done via

```
colloid_cell_list_interactions no
```

The constraint in this case is that $2a_h + h_c < L_{local}$, ie., there is effectively one cell in each local domain. (In practice, the constraint is slightly tighter than this if larger halo regions are in effect.) This option should not be required in normal circumstances.

8.9.2 Initialisation from file

A file containing initial state properties for one or more colloids can be read at run file. This file must be of the correct from, but can be ASCII or binary. To help to create such a file, an example is given in `./util/colloid_file.c`. This file will be read correctly in both serial and in parallel. Colloid positions should always be in terms of the global coordinates. By default, the input expected is `config.cds.init.001-001`. All colloid output is in serial (ie., a single file is produced) at the moment. Again, it may be ASCII or binary.

We present a number of examples taken from the regression tests, the inputs for which are found in

```
trunk/tests/regression
```

For each test there is an input file, and an ASCII file containing the details of the initial colloid state which is read at run time by means of the key value pair

```
colloid_init from_file
```

A file of colloid information may be created with the utility

```
util/colloid_file.c
```

As a minimum, this file must specify:

1. A unique integer id for each particle;
2. a radius and hydrodynamic radius a_0 and a_h (use the same value if unsure);
3. an initial position ($x_{min} < x < x_{max}$) with $x_{min} = 0.5$ and $x_{max} = L_x + 0.5$ etc, where L_x is the appropriate system size for the problem at hand;
4. the initial velocity etc may be safely initialised to zero.

Note that is an inter-particle potential is to be specified at run time, the initial position of the particles should not be so close that a large force is experienced at time $t = 0$. This can destabilise the dynamics.

A single file of colloid information is produced in either ASCII or binary format, which can be read in by specifying the appropriate
`colloid_io_format_input ASCII_SERIAL` or `BINARY_SERIAL`
key value in both serial and parallel.

8.9.3 Very short range potential

See

```
tests/regression/test_spin_solid2_input
tests/regression/config.cds.init.001-001
tests/regression/test_spin_solid2_d3q19.ref*
```

An example of a moderate volume fraction of particles is given for a binary fluid undergoing spinodal decomposition. Here, the capillary interaction between the neutrally wetting colloids causes a strong effective attraction between particles. It is therefore necessary to ensure the particles do not collide to the point of overlapping at their hard-sphere radius. A counterbalancing short-range soft-sphere potential is then defined in the input file:

```
soft_sphere_on 1
```

```
soft_sphere_epsilon 0.0004
```

```
soft_sphere_sigma 0.1
```

```
soft_sphere_nu 1.0
```

```
soft_sphere_cutoff 0.25
```

where, following Eq. 92, we have $v(r) \sim \epsilon(\sigma/r)^\nu$ "cut-and-shifted" so that both potential and force smoothly match to zero at the cut-off distance r_c (here $0.25\Delta x$). The energy scale ϵ will need to be adjusted depending on the exact problem at hand (here it will be related to the fluid-fluid interfacial tension).

Other relevant parameters here are

```
colloid_cell_min 8.0
```

```
lubrication_on 0
```

which control, respectively, the minimum cell list width used to help identify interactions, and the lubrication correction (here switched off).

8.9.4 Short-range potential

See

```
tests/regression/test_yukawa_input
tests/regression/test_yukawa_cds.001-001
tests/regression/test_yukawa_d3q10.ref1
```

This example involves a number of interacting particles in a simple fluid including Brownian motion. The potential is Yukawa-like, representative of a screened Coulomb interaction.

We have the parameters:

```
yukawa_on 1
```

```
yukawa_epsilon 1.330764285
```

```
yukawa_kappa 0.72463768115
```

```
yukawa_cutoff 16.0
```

where the potential is $v(r) \sim \epsilon(-\kappa r)/r$. Again, both potential and force are smoothly matched to zero at the cut-off distance r_c (here $r_c = 16\Delta x$). The relatively long cut-off distance means that the minimum cell list cell width must be correspondingly large (16 lattice units) which limits the minimum parallel domain size.

This example also includes Brownian motion by means of fluctuating hydrodynamics. The relevant parameters in the input are:

```
isothermal_fluctuations on
```

```
temperature 0.0002133333
```

Note that the `temperature` parameter here specifies $k_B T = \langle c_x^2 \rangle = \langle c_y^2 \rangle = \langle c_z^2 \rangle$ in three dimensions (at equilibrium); this is reported in the output of the code at run time.

8.9.5 Long range forces

An Ewald sum is available for magnetic dipoles.

8.9.6 The colloid state

The full data structure for the colloid state is defined in `src/colloid.h` and the corresponding routines which read and write the values are in `src/colloid.c` (note colloid singular!). Most of the state corresponds clearly with physical properties, although there are a number of 'structural' elements which have no physical meaning. These include the `rebuild` flag, which instructs the code to reconstruct the boundary links if necessary, and `deltaphi` which is involved in accounting for conserved order parameter in the `symmetric_lb` free energy. Further, not all the elements are relevant is all circumstances: e.g., the magnetic dipole is only relevant is magnetic interactions are required, and squirmer parameters are only relevant to active particles.

Note that it is possible in principle to have magnetic active particles, in which case the dipole direction (`s`) and the direction of motion vector (`m`) are allowed to be distinct.

8.9.7 Initialisation at random

`colloid_random_no` Sets the total number of particles

`colloid_random_a0`

`colloid_random_ah`

`colloid_random_dh`

The parameter a_0 the nominal radius of the particle on the lattice. This is used to construct the links for BBL. The smallest acceptable LB particle is $a_0 = 1.25$ (approximately). There are 'magic' values including 1.25, 2.3, 3.7, 4.77 which minimise discretisation effects. The hydrodynamic radius of the particles is a_h. For all reasonable applications choose $a_h = a_0$. This is the radius for all physical purposes. The `colloid_random_dh` parameter specifies the smallest allowed surface-surface separation if two or more particles (or walls) are required. This is useful to prevent very close particles experiencing large and destabilising interactions at start time.

The other state for the initial colloids can be set uniformly by means of the keys:

`colloid_random_v0` initial velocity vector

`colloid_random_w0` initial angular velocity vector

`colloid_random_s0` initial spin vector (only relevant in magnetic field)

The

`colloid_random_m0` initial direction of motion vector for active particles

`colloid_random_b1` active parameter b_1

`colloid_random_b2` active parameter b_2 (should be combined with the `active` particle type key as discussed above).

The squirmer parameters specify the surface boundary conditions in an approach originating with Lighthill [30] and Blake [6]. Briefly, b_1 sets the propulsion speed ($U = \frac{2}{3}b_1$), while b_2 sets the particle stresslet. The ratio $\beta = b_2/b_1$ determines the mode of the squirmer motion; see [31] for further details.

There is one special case. If you want exactly one particle, you may set its initial position with

`colloid_random_r0` the initial position vector.

This position must be within the current system, or no particle will be created.

8.9.8 Restarting at random!

Be careful if you are restarting a simulation that used the random initialisation at $t = 0$. You must change to

`colloid_init from_file`

to avoid generating a completely new set of random positions (which will not be consistent with the density and order parameter fields at the restart time). This is almost certainly not what you intended!

8.9.9 Hydrodynamic radius

There is an important issue to understand about the (nominally) spherical particles in LB. The input radius a_0 is used to construct the link boundary conditions. However, discretisation errors in the boundary conditions means that the 'real' particle size may be different. This real radius is the hydrodynamic radius a_h. As a further complication, the hydrodynamic radius can also be a function of viscosity for a given input radius. Some values are given in Table X. This values are computed using a calibration procedure described by Nguyen and Ladd [33].

η	a_0		
	1.25	2.30	4.77
1/6	1.09 (1.05)	2.23 (2.20)	4.76
1/100	1.40 (1.34)	2.50 (2.46)	5.01
1/1000	1.63	2.71	5.23

Table 5: A table of calibrated hydrodynamic radii as a function of input radius and fluid viscosity η. These figures are for D3Q19 (and D3Q15 in brackets).

If a calibration is required for a given radius and viscosity, use the `calibration on` key pair in the input file. Some examples are given in the `tests/calibration` directory. An appppropriate number of time steps must be used to allow the system to get over the initial transient (we use the momentum diffusion time for the system L^2/η) and to allow at least one particle Stokes time (a/U). The input radius and hydrodynamic radius should be set the same in the input file.

8.9.10 Flat wall lubrication correction

It is possible to prevent colloids colliding with the flat boundary walls by adding the correction to the normal component of the lubrication force. The general form of this correction is

$$\mathbf{F}_{\text{lub}} = -6\pi\eta a_h^2(1/h - 1/h_c)\mathbf{u}.\mathbf{r}. \tag{103}$$

Here, h is the surface to surface separation, and h_c is the cutoff distance beyond which the correction is not applied. The relative velocity (assuming the wall is not moving) and normal separation are $\mathbf{u}$ and $\mathbf{r}$ respectively.

The cutoff distance can be set from the input via

```
boundary_lubrication_rcnormal 0.1
```

where the value is (strictly) set via calibration, cf. the hydrodynamic radius. In general, the cutoff is between zero and 0.5 lattice spacing, and can be smaller for larger particles. Because the wall velocity is fixed, this force correction should cause no stability issues in the colloid velocity update (provided a colloid's initial position is not very close to the wall). By default the cutoff $h_c = 0$.

9 Electrokinetics

9.1 Sress tensor in the electrosymmetic model

9.1.1 Definitions

The total free energy functional is a function of the composition ϕ, the density of the ionic species ρ_α,

$$\mathcal{F}[\phi, \{\rho_\alpha\}] = \int d\mathbf{r}\; F\left(\phi(\mathbf{r}), \{\rho_\alpha(\mathbf{r})\}\right). \tag{104}$$

The free energy density can be factorised into a contribution from the composition and an ionic contribution according to

$$\begin{aligned}
F &= F^{mix} + F^{ion} && (105)\\
F^{mix} &= \frac{1}{2} A\,\phi^2 + \frac{1}{4} B\,\phi^4 + \frac{1}{2}\kappa(\boldsymbol{\nabla}\phi)^2 && (106)\\
F^{ion,ex} &= \sum_{\alpha=\pm} \rho_\alpha \left(V_\alpha^{solv} - \mu_\alpha + \frac{z_\alpha e}{2}\Psi \right). && (107)
\end{aligned}$$

In the ionic contribution we have omitted the ideal gas contribution of the ions. The solvation energy is given by

$$V_\alpha^{solv}(\mathbf{r}) = \Delta\mu_\alpha \frac{1+\phi(\mathbf{r})}{2} \qquad (108)$$

It is convenient to define the total free energy without electric field F_0, i.e. with vanishing electric potential:

$$F_0 \quad = \quad F^{mix} + F^{ion,ex}|_{\Psi=0} \qquad (109)$$

The Poisson equation reads

$$\boldsymbol{\nabla}\left(\varepsilon(\mathbf{r})\boldsymbol{\nabla}\Psi(\mathbf{r})\right) = -\sum_{\alpha=\pm} e\, z_\alpha\, \rho_\alpha(\mathbf{r}) \qquad (110)$$

whereas the electric field is given by

$$\boldsymbol{E} = -\boldsymbol{\nabla}\Psi. \qquad (111)$$

The permittivity depends on the composition ϕ via

$$\varepsilon(\mathbf{r}) = \bar{\varepsilon}\,(1 - \gamma\,\phi(\mathbf{r})) \qquad (112)$$

The electric displacement field

$$\boldsymbol{D}(\mathbf{r}) = \varepsilon(\mathbf{r})\boldsymbol{E}(\mathbf{r}) \qquad (113)$$

The 'co-energy' density

$$\tilde{F} = F_0 - \frac{\varepsilon}{2}\boldsymbol{E}^2 \qquad (114)$$

has the meaning of a Legendre transform of the free energy density F_0.

9.1.2 Electromechanical stress tensor

The electromechanical stress tensor is generally defined as [48, 49].

$$\sigma_{ij} = \varepsilon E_i E_j + \delta_{ij}\left(\tilde{F} - \phi\frac{\delta\tilde{\mathcal{F}}}{\delta\phi} - \sum_{\alpha=\pm}\rho_\alpha\frac{\delta\tilde{\mathcal{F}}}{\delta\rho_\alpha}\right). \tag{115}$$

The first functional derivative yields

$$\begin{aligned}
\phi\frac{\delta\tilde{\mathcal{F}}}{\delta\phi} &= \phi\left(\frac{\delta\mathcal{F}_0}{\delta\phi} - \frac{\boldsymbol{E}^2}{2}\frac{\partial\varepsilon}{\partial\phi}\right) && (116)\\
&= \phi\left(\frac{\partial F_0}{\partial\phi} - \boldsymbol{\nabla}\left(\frac{\partial F_0}{\partial\boldsymbol{\nabla}\phi}\right) - \frac{\boldsymbol{E}^2}{2}\frac{\partial\varepsilon}{\partial\phi}\right) && (117)\\
&= A\,\phi^2 + B\,\phi^4 - \kappa\phi(\boldsymbol{\nabla}^2\phi) + \frac{\phi}{2}\sum_{\alpha=\pm}\rho_\alpha\Delta\mu_\alpha + \phi\,\frac{\bar{\varepsilon}\gamma}{2}\boldsymbol{E}^2 && (118)\\
&= A\,\phi^2 + B\,\phi^4 - \kappa\phi(\boldsymbol{\nabla}^2\phi) + \frac{\phi}{2}\sum_{\alpha=\pm}\rho_\alpha\Delta\mu_\alpha - \frac{\varepsilon-\bar{\varepsilon}}{2}\boldsymbol{E}^2, && (119)
\end{aligned}$$

where we used the above dependence of the permittivity on the composition. The second functional derivative is given by

$$\begin{aligned}
\rho_\alpha\frac{\delta\tilde{\mathcal{F}}}{\delta\rho_\alpha} &= \rho_\alpha\left(\frac{\delta\mathcal{F}_0}{\delta\rho_\alpha}\right) && (120)\\
&= \rho_\alpha(V_\alpha^{solv} - \mu_\alpha) && (121)
\end{aligned}$$

The co-energy density and the functional derivatives with respect to composition ϕ and ionic densities ρ_α add up to

$$\begin{aligned}
&\tilde{F} - \phi\frac{\delta\tilde{F}}{\delta\phi} - \sum_{\alpha=\pm}\rho_\alpha\frac{\delta\tilde{F}}{\delta\rho_\alpha} =\\
&\quad= -\frac{\varepsilon}{2}\boldsymbol{E}^2 + \frac{A}{2}\phi^2 + \frac{B}{4}\phi^4 + \frac{\kappa}{2}(\boldsymbol{\nabla}\phi)^2\\
&\qquad -A\phi^2 - B\phi^4 + \kappa\phi(\boldsymbol{\nabla}^2\phi) - \frac{\phi}{2}\sum_{\alpha=\pm}\rho_\alpha\Delta\mu_\alpha\\
&\qquad -\phi\,\frac{\bar{\varepsilon}\gamma}{2}\boldsymbol{E}^2 - \sum_{\alpha=\pm}\rho_\alpha(V_\alpha^{solv} - \mu_\alpha) && (122)\\
&\quad= -\frac{\varepsilon}{2}\boldsymbol{E}^2 - \frac{A}{2}\phi^2 - \frac{3B}{4}\phi^4 + \frac{\kappa}{2}(\boldsymbol{\nabla}\phi)^2 + \kappa\phi(\boldsymbol{\nabla}^2\phi)\\
&\qquad -\frac{\phi}{2}\sum_{\alpha=\pm}\rho_\alpha\Delta\mu_\alpha - \phi\,\frac{\bar{\varepsilon}\gamma}{2}\boldsymbol{E}^2 && (123)
\end{aligned}$$

This yields

$$\begin{aligned}\sigma_{ij} &= \varepsilon E_i E_j - \delta_{ij}\Bigg(\frac{\varepsilon}{2}\boldsymbol{E}^2 + \frac{A}{2}\phi^2 + \frac{3\,B}{4}\phi^4 - \frac{\kappa}{2}(\boldsymbol{\nabla}\phi)^2 \\ &\quad -\kappa\phi(\boldsymbol{\nabla}^2\phi) + \frac{\phi}{2}\sum_{\alpha=\pm}\rho_\alpha\Delta\mu_\alpha + \phi\,\frac{\bar{\varepsilon}\gamma}{2}\boldsymbol{E}^2\Bigg). \end{aligned} \tag{124}$$

In order to satisfy the condition of mechanical equilibrium $\nabla_j\sigma_{ij} = 0$ a symmetric contribution has to be added to the above stress tensor. The following lines try to elucidate this.

Taking the divergence of the stress tensor gives

$$\nabla_j\sigma_{ij} = (\nabla_j\varepsilon E_j)E_i + \varepsilon E_j\nabla_j E_i + \nabla_i\left(\tilde{F} - \phi\frac{\delta\tilde{\mathcal{F}}}{\delta\phi} - \sum_{\alpha=\pm}\rho_\alpha\frac{\delta\tilde{\mathcal{F}}}{\delta\rho_\alpha}\right) \tag{125}$$

$$\begin{aligned} &= (\boldsymbol{\nabla}\cdot\boldsymbol{D})\,E_i + D_j\nabla_j E_i + \nabla_i F_0 - D_j\nabla_i E_j \\ &\quad -\nabla_i\left(\phi\frac{\delta\tilde{\mathcal{F}}}{\delta\phi} + \sum_{\alpha=\pm}\rho_\alpha\frac{\delta\tilde{\mathcal{F}}}{\delta\rho_\alpha}\right) \end{aligned} \tag{126}$$

$$\begin{aligned} &= (\boldsymbol{\nabla}\cdot\boldsymbol{D})\,E_i + D_j\left(\nabla_j E_i - \nabla_i E_j\right) + \nabla_i F_0 \\ &\quad -\nabla_i\left(\phi\mu_\phi + \sum_{\alpha=\pm}\rho_\alpha\mu_{\rho_\alpha}\right) \end{aligned} \tag{127}$$

According to Gauss' law the first term is

$$(\boldsymbol{\nabla}\cdot\boldsymbol{D})\,E_i = \rho_f E_i = \sum_{\alpha=\pm} z_\alpha e\rho_\alpha E_i, \tag{128}$$

whereas ρ_f is the free charge density. Moreover, due to Faraday's law the curl in the second bracket vanishes. This leads us to the following intermediate result:

$$\nabla_j\sigma_{ij} = \sum_{\alpha=\pm} z_\alpha e\rho_\alpha E_i + \nabla_i F_0 - \nabla_i\left(\phi\mu_\phi + \sum_{\alpha=\pm}\rho_\alpha\mu_{\rho_\alpha}\right) \tag{129}$$

$$\begin{aligned} &= \sum_{\alpha=\pm} z_\alpha e\rho_\alpha E_i + \frac{\partial F_0}{\partial\phi}\nabla_i\phi + \frac{\partial F_0}{\partial\nabla_j\phi}\nabla_j\nabla_i\phi + \sum_{\alpha=\pm}\left\{\frac{\partial F_0}{\partial\rho_\alpha}\right\}\nabla_i\rho_\alpha \\ &\quad -\nabla_i\left(\phi\mu_\phi + \sum_{\alpha=\pm}\rho_\alpha\mu_{\rho_\alpha}\right) \end{aligned} \tag{130}$$

$$\begin{aligned} &= \sum_{\alpha=\pm} z_\alpha e\rho_\alpha E_i + \left\{\frac{\partial F_0}{\partial\phi} - \nabla_j\left(\frac{\partial F_0}{\partial\nabla_j\phi}\right)\right\}\nabla_i\phi + \nabla_j\left(\frac{\partial F_0}{\partial\nabla_j\phi}\nabla_i\phi\right) \\ &\quad + \sum_{\alpha=\pm}\left\{\frac{\partial F_0}{\partial\rho_\alpha}\right\}\nabla_i\rho_\alpha - \nabla_i\left(\phi\mu_\phi + \sum_{\alpha=\pm}\rho_\alpha\mu_{\rho_\alpha}\right). \end{aligned} \tag{131}$$

If we replace the terms in curly brackets the chemical potentials μ_ϕ and μ_{ρ_α} and use the fact that gradients of the chemical potential vanish in equilibrium, we end up with

$$\nabla_j \sigma_{ij} \;=\; \sum_{\alpha=\pm} z_\alpha e \rho_\alpha E_i + \nabla_j \left(\frac{\partial F_0}{\partial \nabla_j \phi} \nabla_i \phi \right) \tag{132}$$

The first term is the external force on the charges, which is zero if we have no counterions and an equal amount of positve and negative ions. Therefore, we have to add a term

$$-\left(\frac{\partial F_0}{\partial \nabla_j \phi} \nabla_i \phi \right) \tag{133}$$

to the stress tensor to fulfill the equilibrium condition. This is equivalent to similar derivations in [50].

The total stress tensor reads

$$\begin{aligned} \sigma_{ij} \;=\; & \varepsilon E_i E_j - \kappa (\nabla_i \phi)(\nabla_j \phi) - \delta_{ij} \left(\frac{\varepsilon}{2} \boldsymbol{E}^2 + \frac{A}{2} \phi^2 + \frac{3\,B}{4} \phi^4 \right. \\ & \left. - \frac{\kappa}{2} (\boldsymbol{\nabla} \phi)^2 - \kappa \phi (\boldsymbol{\nabla}^2 \phi) + \frac{\phi}{2} \sum_{\alpha=\pm} \rho_\alpha \Delta \mu_\alpha + \phi \, \frac{\bar{\varepsilon} \gamma}{2} \boldsymbol{E}^2 \right). \end{aligned} \tag{134}$$

9.2 Examples

A number of regression tests verify the implemention of the Nernst-Planck equation in combination with the SOR solver. They can be found in

```
trunk/tests/regression
```

The original tests we carried out during the first implementation comprise the Gouy-Chapman theory for electric double layers in front of a charged wall [44], the liquid junction potential emerging between two electrolytes of slighty different concentration and diffusivity of the charged species [45], electro-osmotic flow in a slit pore [46, 47] and Debye-Hückel theory for charged colloidal particles and small enough potentials [44]. They are described in the following paragraphs.

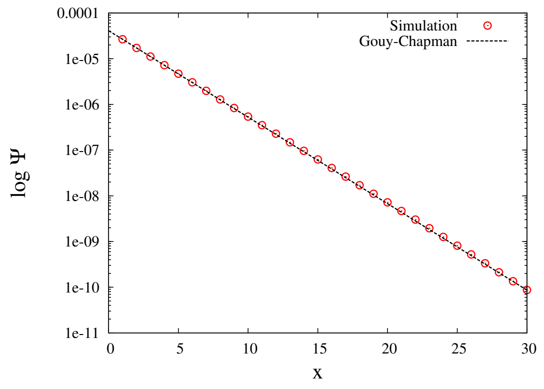
9.2.1 Gouy-Chapman

This validation test is a flat surface carrying a surface charge σ with counterions and symmetic electrolyte. This is a one-dimensional, electroneutral problem of a diffusive electric double layer which has an analytical solution [44]. The approximation for low electrostatic potentials reads

$$\Psi(x) = \Psi_D \exp(-\kappa\, x) \tag{135}$$

with κ as inverse Debye length

$$\kappa = l_D^{-1} = \sqrt{8\pi \, l_B \, I}. \tag{136}$$



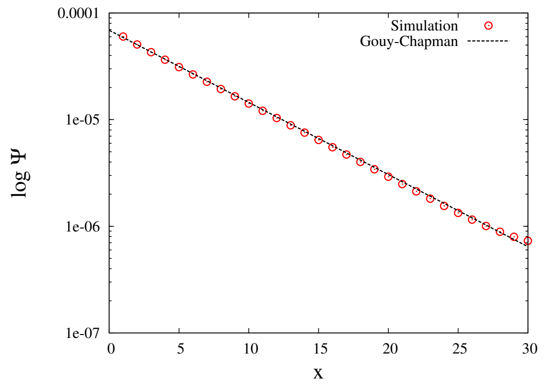


Figure 3: Gouy-Chapman theory for electric double layers: The plots compare the simulation results for the electric potential with the analytical solution for $\rho_{0,\pm} = 1 \cdot 10^{-2}$ (left) and $\rho_{0,\pm} = 1 \cdot 10^{-3}$ (right).

The Bjerrum length l_B is given by

$$l_B = \frac{\beta\, e^2}{4\pi\, \varepsilon} \tag{137}$$

with $\beta^{-1} = k_B T$, e as unit charge and $\varepsilon = \varepsilon_0 \varepsilon_r$ as dielectric permittivity. The parameter

$$I = \frac{1}{2} \sum_k z_k^2\, \rho_{B,k} \tag{138}$$

is the ionic strength of the electrolyte with z_k as valencies of species k ($z_\pm = \pm 1$ for simple symmetric electrolyte) and $\rho_{B,k}$ as bulk charge density of species k far away from the wall. The Stern potential Ψ_D at the surface of the wall is related to the surface charge σ via

$$\Psi_D = \frac{2}{\beta\, e} \sinh^{-1}(-\sigma\, p) \tag{139}$$

$$\Psi_D \simeq \frac{2}{\beta\, e} \ln\left(-\sigma\, p + \sqrt{(\sigma\, p)^2 + 1}\right) \tag{140}$$

$$p = \frac{1}{\sqrt{8\, \varepsilon\, \beta^{-1}\, \rho_B}}. \tag{141}$$

The quantity $\rho_B = \rho_{B,+} = \rho_{B,-}$ is the average bulk charge density of the electrolyte.

We solved the Gouy-Chapman problem for a system consisting of $L_x \times L_y \times L_z = 64 \times 4 \times 4$ lattice sites. Periodic boundary conditions were used at all sides and a no-flux boundary conditions was set at $L_x = 1$ and $L_x = 64$.

The parameters were chosen as follows (all in simulation units): unit charge $e = 1$, temperature $k_B T = \beta^{-1} = 3.333 \cdot 10^{-5}$, valency $z_\pm = \pm 1$, dielectric permittivity $\varepsilon = 3.3 \cdot 10^3$, diffusivities of the species $D_\pm = 1 \cdot 10^{-2}$ and initial densities $\rho_{0,\pm} = 1 \cdot 10^{-2}$. The entire system was electro-neutral and had a Bjerrum length $l_B = 0.723$.

At first we tested the linear case applying a small positive surface charge $\sigma = 3.125 \cdot 10^{-2}$, which led to bulk charge density $\rho_{B,+} = \rho_{B,-} = 1.044 \cdot 10^{-2}$, Debye length $l_D = 2.295$

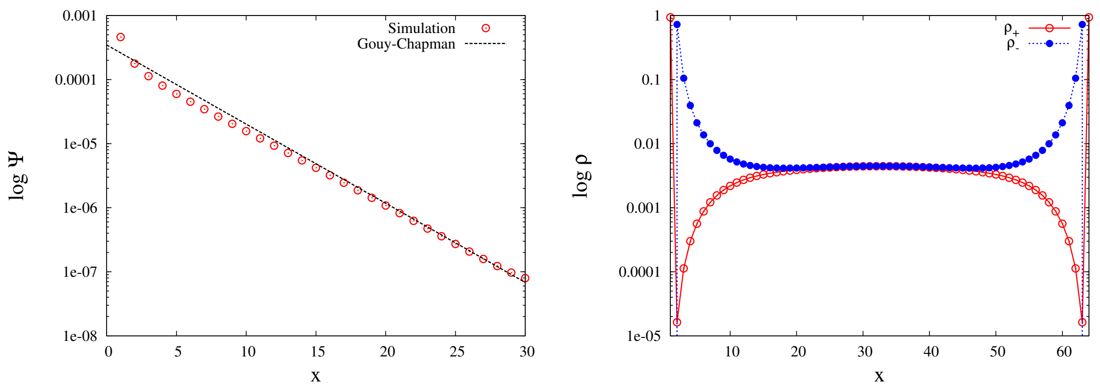


Figure 4: Nonlinear regime: For larger surface charges the solution deviates from the low-potential approximation Eq. 135 to the Gouy-Chapman theory. The righthand picture shows the charge distribution across the gap.

and surface potential $\Psi_D = 2.136 \cdot 10^{-5}$. The potential was initialised with zero and had a value $\Psi_c = -2.364 \cdot 10^{-6}$ in the centre of the system after equilibration which we subtracted in the following analysis.

We reduced the initial density of the electrolyte to $\rho_{0,\pm} = 1 \cdot 10^{-3}$, which resulted in bulk charge densities $\rho_{B,+} = 1.298 \cdot 10^{-3}$, $\rho_{B,-} = 1.370 \cdot 10^{-3}$, Debye length $l_D = 6.420$, surface potential $\Psi_D = 5.451 \cdot 10^{-5}$ and centre potential $\Psi_c = -1.256 \cdot 10^{-5}$. The comparison with the approximate solution Eq. 135 is shown in Fig. 3.

For larger surface charges the low-potential assumption which led to Eq. 135 is no longer valid and the nonlinear nature of the Poisson-Boltzmann equation becomes evident. Fig. 4 shows the results for a surface charge $\sigma = 9.375 \cdot 10^{-1}$ and electrolyte density $\rho_{0,\pm} = 3 \cdot 10^{-3}$. We obtained for bulk charge densities $\rho_{B,+} = 4.443 \cdot 10^{-3}$ and $\rho_{B,-} = 4.461 \cdot 10^{-3}$, Debye length $l_D = 3.514$, the surface potential $\Psi_D = 2.267 \cdot 10^{-4}$ and the centre potential $\Psi_c = -3.395 \cdot 10^{-5}$.

9.2.2 Liquid-junction potential

The liquid junction potential is a charge separation process that occurs when electrolytes with slightly different concentrations whose species have different diffusivities are brought into contact. Charges from the regions of higher concentration diffuse into the parts with lower concentration. Due to the difference in diffusivity they migrate at different speeds, leaving parts of the system charged. This leads to a build-up of a potential which balances the diffusive flux.

After the initial build-up phase the potential decreases slowly again until the charge concentration has become homogeneous throughout the system. Both timescales of emergence and decay of the potential can be separated by chosing a sufficiently large system size.

This problem allowed us to verify the correct temporal behaviour of the Nernst-Planck equation solver by resolving the transient dynamics without having to account for advective terms.

For simplicity we considered systems of size $L_x \times L_y \times Lz = 128 \times 4 \times 4$ and $L_x \times L_y \times Lz = 256 \times 4 \times 4$ with periodic boundary conditions at either end. The two halfs were

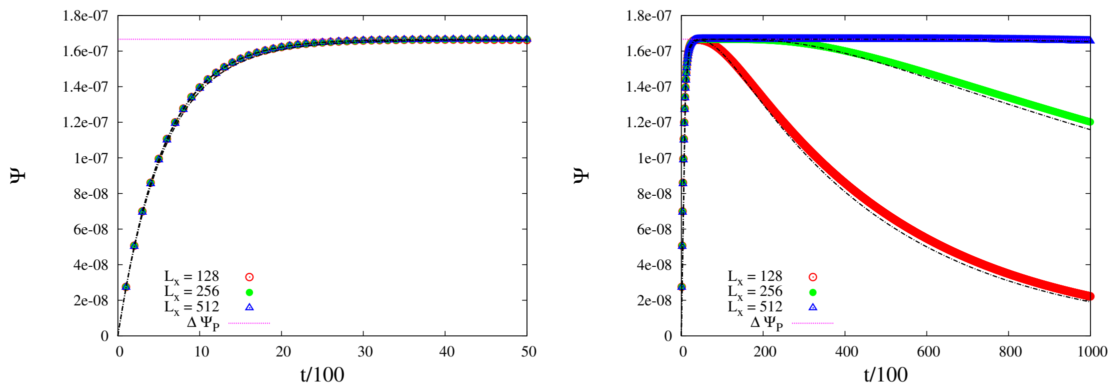


Figure 5: Time evolution of the liquid junction potential for $L_x = 128$ (blue), $L_x = 256$ (green) and $L_x = 512$ (blue). The dashed black curves represent the approximate solution in the limit $l_D/L_x << 1$. Deviations can be seen which are presumably due to the approximate nature of the analytical solution and the fact that we have a different asymptotic flux close to the boundary - the theoretical analysis assumes no flux at a boundary which is infinitely far away from the diffusive region.

electroneutral and had ionic concentrations $\rho_{L,\pm} = \rho_{0,\pm} + \delta\rho$ and $\rho_{R,\pm} = \rho_{0,\pm} - \delta\rho$ with $\rho_{0,\pm} = 1 \cdot 10^{-2}$ and $\delta\rho = 0.01$.

The potential difference between both sides during the build-up obeys approximately

$$\Delta\Psi(t) \simeq \Delta\Psi_P \left\{ 1 - \exp\left(-\frac{t}{\tau_e} \right) \right\} \tag{142}$$

with

$$\Delta\Psi_P = \frac{(D_+ D_-)}{\beta e (D_+ + D_-)} \frac{\delta\rho}{\rho_0} \tag{143}$$

as saturation value of the potential difference. The saturation time scale is given by

$$\tau_e = \frac{\varepsilon}{\beta\, e^2 (D_+ + D_-) \rho_0}. \tag{144}$$

A more exact solution can be derived in the limit $l_D/L_x << 1, N \to \infty$:

$$\begin{aligned} \Delta\Psi(t) &= \Delta\Psi_P \left\{ 1 - \exp\left(-\frac{t}{\tau_e} \right) \right\} \frac{4}{\pi} \left\{ \sum_{m=1}^{N} \frac{\sin^3(m\pi/2)}{m} \exp\left(-\frac{m^2\, t}{\tau_d} \right) \right\} & (145) \\ \tau_d &= \frac{L^2}{2\pi^2 (D_+ + D_-)}. & (146) \end{aligned}$$

It contains as well the dependence on the decay timescale τ_d. Only odd indices m contribute to the sum:

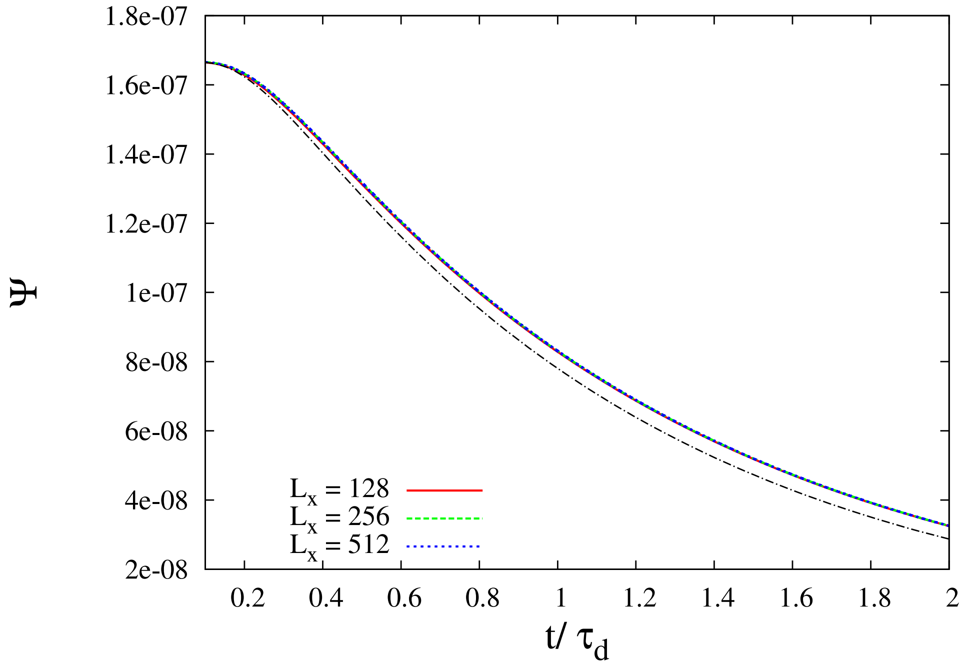


Figure 6: Rescaled plot of the decay: Times have been Time evolution of the liquid junction potential for $L_x = 128$ (blue), $L_x = 256$ (green) and $L_x = 512$ (blue). The dashed black curves represent the approximate solution in the limit $l_D/L_x << 1$. Deviations can be seen which are due to the approximate nature of the analytical solution and the diffusive fluxes close to the boundary.

$$\begin{aligned}\sum_{m=1}^{N} \frac{\sin^3(m\pi/2)}{m} \exp\left(-\frac{m^2\, t}{\tau_d}\right) &= \\ \exp\left(-\frac{t}{\tau_d}\right) - \frac{1}{3}\exp\left(-\frac{9t}{\tau_d}\right) + \frac{1}{5}\exp\left(-\frac{25t}{\tau_d}\right) - \frac{1}{9}\exp\left(-\frac{81t}{\tau_d}\right) &+ \ldots \end{aligned} \tag{147}$$

A complete discussion of the solution can be found in [45]. There, the upper limit of significant modes has been also estimated as $N_{max} = L/\pi l_D$. Note the factor 2 difference between Eq. 146 and the corresponding expression in [45].

The following parameters were used: dielectric permittivity $\epsilon = 3.3 \cdot 10^3$, temperature $\beta^{-1} = 3.333 \cdot 10^{-5}$, unit charge $e = 1$, valency $z_\pm = \pm 1$, diffusivities $D_+ = 0.0125$ and $D_- = 0.0075$. We obtained $\Delta\Psi_P = 1.6667 \cdot 10^{-7}$, $\tau_e = 550$, $\tau_d = 41501.2\,(L_x = 128)$, $\tau_d = 166004.6\,(L_x = 256)$ and $\tau_d = 664018.5(L_x = 512)$. The results for the potential difference over time are shown in Fig. 5.

Fig. 6 shows results with times rescaled to the decay time scale τ_d (cf. Eq. 146). Obviously the deviations we observe are not due to the limited system size and have a more systematic origin.

The curves coincide if the theoretic limit for τ_d is rescaled by a factor 1.067, suggesting the effective system length for this sort of setup is actually about 3% larger than the numerical value.

A reason for this might be the approximate nature of the analytical solution and the fact that it was gained for an infinitely large system with constant charge concentrations, vanishing currents at both ends and finite diffusive zone of size L_x. In our situation the entire system is within the diffusive zone. This may lead to smaller effective diffusivities or larger effective system sizes. Interestingly, all runs with solid walls at both ends resulted in oscillatory behaviour and an effective system size of $2L_x$.

9.2.3 Electroosmotic Flow

To test the implementation with all couplings to external and internal forces we consider a forced charged fluid in a slit of size L_z. An electrostatic field $E_{||}$ is allied parallel to the walls. The entire system is electroneutral with each wall having the surface charge density σ and compensationg counterions with total charge $2\sigma A_{wall}$ in the fluid.

In equilibrium the charge density at a distance x from the wall obeys

$$\rho(x) = \frac{\rho_0}{\cos^2(K\,x)} \tag{148}$$

with

$$\rho_0 = K^2/2\pi l_B \tag{149}$$

and

$$K\,L_x \tan\left(\frac{K\,L_x}{2}\right) = \pi\, l_B\, L_x\, 2\sigma \tag{150}$$

$$K\,L_x \simeq \sqrt{4\pi\, l_B\, L_x\, 2\sigma}. \tag{151}$$

The liniarised version Eq. 151 has only a limited range of applicabilty. We solved Eq. 150 numerically and found solutions $K = 0.01959$ ($\sigma = 0.003125$) and $K = 0.03311$ ($\sigma = 0.00125$), which is reasonably far away from the theoretical limit $K_{max} = \pi/L_x$ set by the tangent.

Note the factor 2 difference on the lhs of Eq. 150 with respect to [46, 47]. There is also a factor L_x missing on the rhs of Eq. 151.

The steady state velocity of the fluid can be derived from the force balance of the gradient of the stresses and the electrostatic forces:

$$v_y(x) = \hat{v} \ln \left(\frac{\cos(K\,x)}{\cos(K\,L_x/2)} \right) \tag{152}$$

$$\hat{v} = \frac{e\,E_{||}\rho_o}{\eta\,K^2} = \frac{e\,E_{||}}{2\pi\eta l_B} \tag{153}$$

The result for two different charge densities is shown in Fig. 10. The accuracy is acceptable with deviations for high surface charged potentially being caused by the chosen discretisation or by the numerical solution of Eq. 150 approaching the limit of $\pi/L_z \simeq 0.049$.

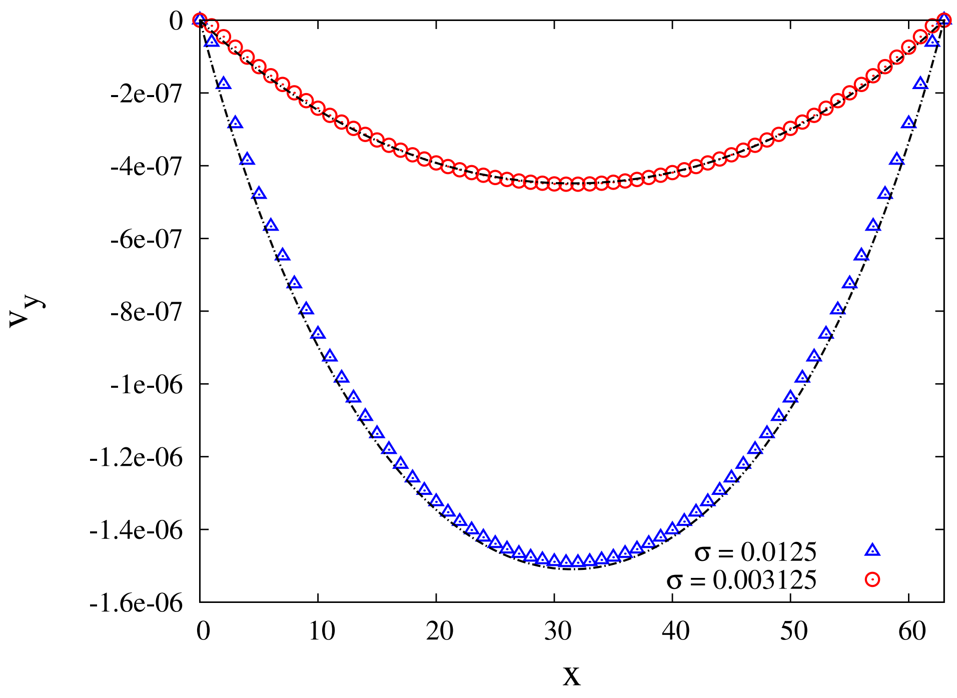


Figure 7: Steady state flow profile across a slit of width $L_x = 63$ for an applied field of magnitude $e\beta E L_x = 1.89$ for two different charge densities $\sigma = 0.003125$ (red) and $\sigma = 0.0125$ (blue), Bjerrum length $l_B = 0.7234$, viscosity $\eta = 0.1$ and unit charge $e = 1$. The dashed black lines correspond to theoretical prediction according to Eqs. 152 and 153.

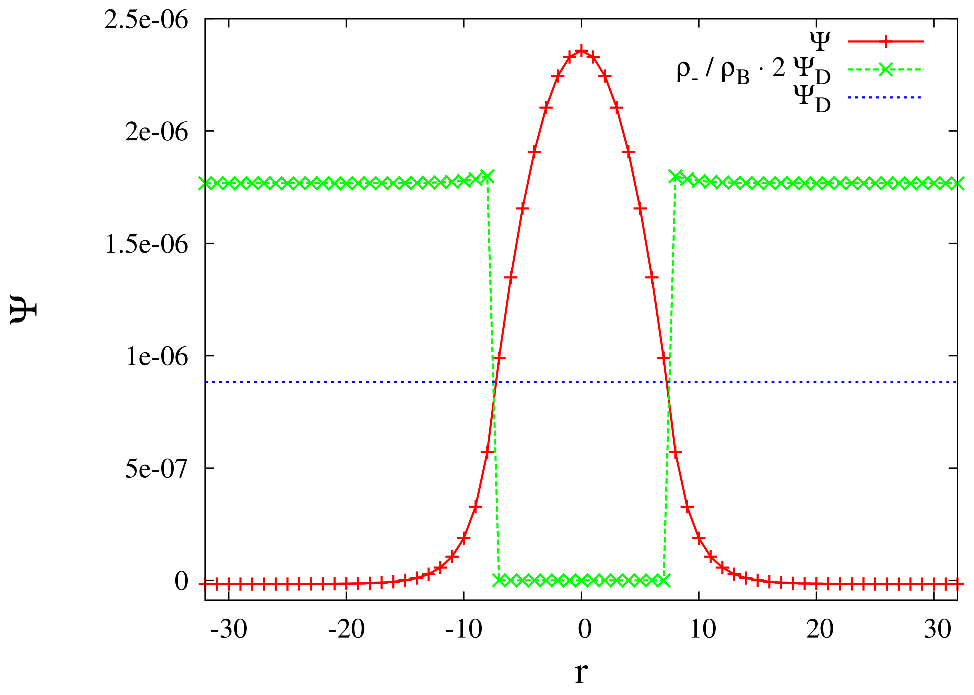
9.2.4 Debye-Hückel Theory

We have tested the implementation for a single fixed colloid and compared the result with Debye-Hückel theory. A result is shown in Fig. 8. We used the following parametrisation:

$L_x \times L_y \times L_z = 64 \times 64 \times 64$, $D_+ = D_- = 0.01$, $e = 1$, $z_\pm = 1$, $\beta^{-1} = 3.333 \cdot 10^{-5}$, $\varepsilon = 3.3 \cdot 10^3$, $l_B = 0.723$.

For a central and fixed colloid of radius $R_c = 7.5$ carrying a positive unit charge $q_{c,+} = 1.0, q_{c,-} = 0$ we obtained $\rho_{c,+} = 5.58 \cdot 10^{-4}$, $\rho_{el} = \rho_{B,\pm} = 1 \cdot 10^{-2}$, $\Psi_D = 8.836 \cdot 10^{-7}$ for $2\,R_c = 16$.

For a higher larger positive charge $q_{c,+} = 4.0$ we got $\rho_{c,+} = 2.23 \cdot 10^{-3}$ $\rho_{el} = \rho_{B,\pm} = 5 \cdot 10^{-3}$ $\Psi_D = 4.993 \cdot 10^{-6}$ for $2\,R_c = 16$.



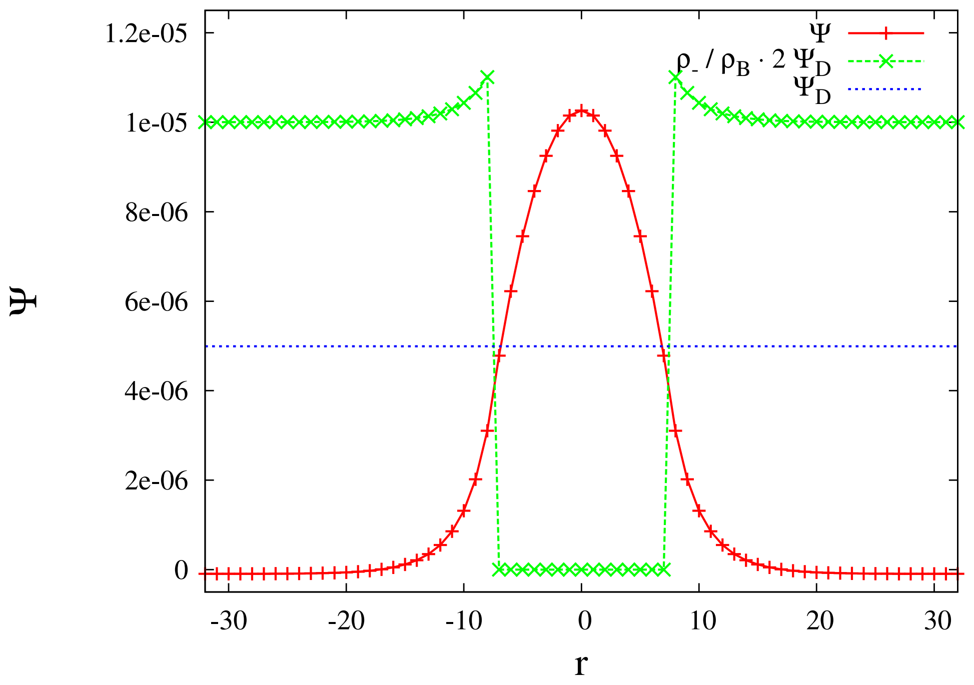


Figure 8: Debye-Hückel theory for positively charged colloid: the picture shows a cut through the center of a single colloid in a peridoic system. The potential is shown in red, the Stern potential in blue, and the negative charge density in green. The colloid is in the center.

9.2.5 Scaling of the SOR

We have run some simple scaling tests for the electrokinetic problem with zero fluid velocity. A system of 128^3 lattice sites was used as representative of a small production system. The code was run on the Mare Nostrum machine in Barcelona for a small number of time steps to assess performance on up to 512 MPI tasks. The results are shown in Fig. 9.

While the scaling for the Nernst Planck part of the calulcation are reasonable, it can be seen that the SOR routine used to solve the Poisson equation is not really acceptable. This was not unexpected. Some improvements might be possible, but the algorithm is essentailly unsuitable for large systems.

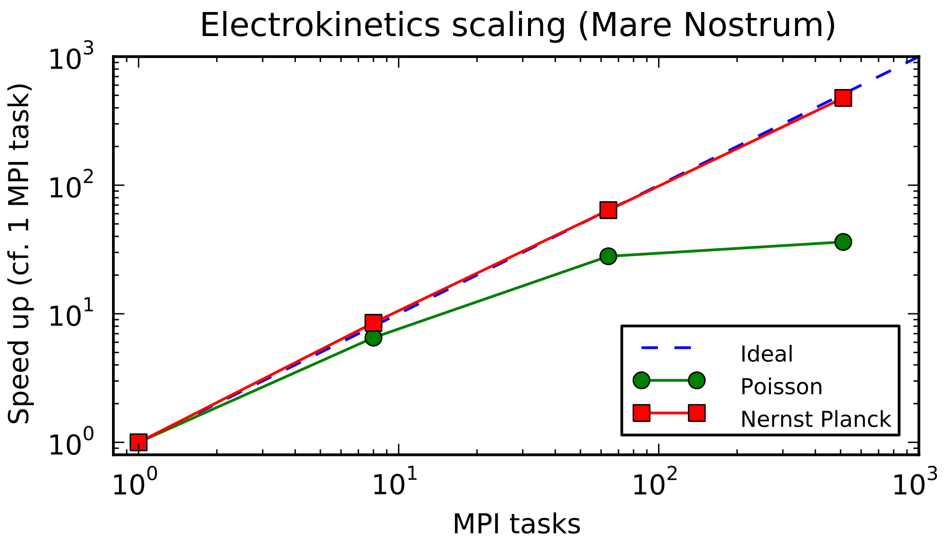


Figure 9: Scaling of a system of 128^3 lattice sites on up to 512 MPI tasks. While the scaling for the Nernst Planck part of the calulcation are reasonable, it can be seen that the SOR routine used to solve the Poisson equation deviates strongly from ideal behaviour.

9.2.6 Scaling with Krylov Subspace Solver

We use a Krylov subspace solver from the Portable Extendable Toolkit for Scientific Computation (PETSc).

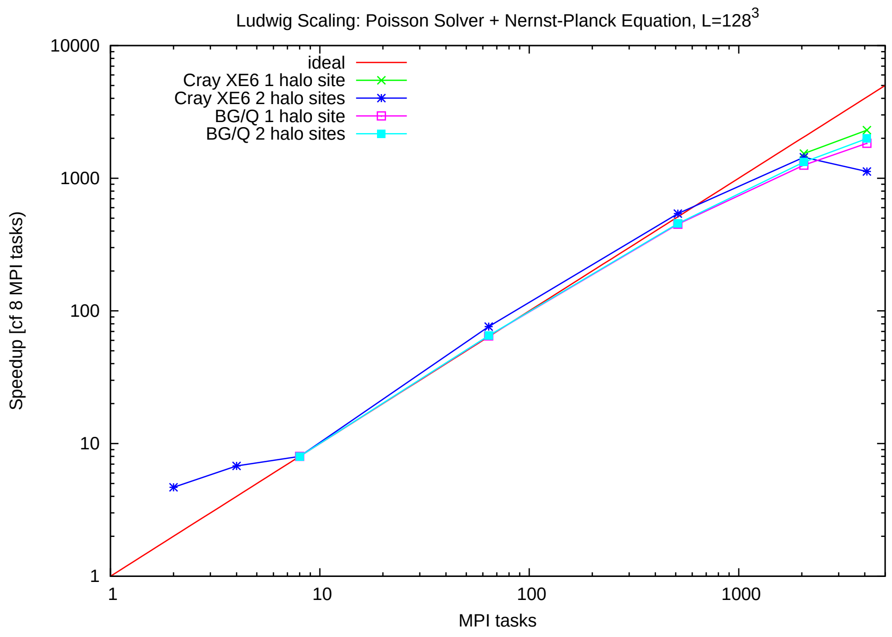


Figure 10: Scaling of a system of 128^3 lattice sites on up to 4096 MPI tasks. The scaling of both the Nernst Planck part as well as the Poisson solver are only limited by the overhead of communication to computation at very small decompositions.

9.3 User input

10 Software Configuration Management Plan

10.1 Introduction

10.1.1 Purpose

This section contains information on the *Software Configuration Management Plan* for the *Ludwig* code. It is to provide users of the code a background on the way in which the code is developed, how bugs are dealt with, the way in which new features may be added, how correctness is ensured and tested, and so on. It is therefore a part of the process by which the code is maintained. The Software Configuration Management Plan will be referred to as 'The Plan' throughout the remainder of the section.

10.1.2 Scope

The *Ludwig* code is designed to study simple and complex fluids based around the numerical solution of the Navier-Stokes equations. Components of *Ludwig* include the main program, unit and regression tests, and a small number support libraries, and a small number of utility programs used to prepare input and post-process output. The Plan is limited to these software components.

10.1.3 Relationship to organisation and other projects

Ludwig is an on-going research project, and relies mainly on funding for the United Kingdom Engineering and Physical Sciences Research Council to provide support for staff time to work on maintenance and development.

10.1.4 References

This section is based on the IEEE standard for Configuration Management IEEE 828-2012 [21], and follows the format set out therein. Relevant references are included and can be found in the main References section at the end of the document.

10.2 Identification

The material under control of the project includes a number of sets of files: this documentation, the source code, the build and test suite which is used to monitor the status of the code, and so on. Control of project files is via a single on-line repository which uses the subversion (SVN) revision control system. The SVN repository is currently located at

`http://ccpforge.cse.rl.ac.uk/`

which is maintained at the Rutherford Appleton Laboratory on behalf of Collaborative Computational Physics 5 (The Computer Simulation of Condensed Phases). Subversion identifies different revisions by a unique revision number. A given version of a file is then uniquely identified by its path in the repository as it exists at a given revision number.

Specifically, configuration items are taken to include files in the trunk of the repository

`http://ccpforge.cse.rl.ac.uk/svn/ludwig/trunk`

Other branches in the repository are not considered to be under configuration management.

10.3 Responsibility and authority

While the Soft Matter Physics Group is responsible for the overall scientific direction and development of the code and concomitant priorities, all Configuration Management activities will be the responsibility of EPCC at The University of Edinburgh.

10.4 Project organisation

10.4.1 Organisations

Ludwig is developed by a team based in the School of Physics at The University of Edinburgh. This involves two groups: the Soft Matter Physics Group, and Edinburgh Parallel Computing Centre (EPCC). These groups collaborate with a number of workers at different Universities around the world on different aspects of the code.

10.4.2 Process Management

EPCC is responsible for the Software Configuration Management process. It is anticipated that the cost of this process will be small as the project team can communicate informally on a regular basis. There is currently no independent surveillance of activities to ensure compliance with The Plan.

10.4.3 Control

Requests for changes or additions to the code should be made via the issue tracker at CCPForge. The project team will then decide whether the change is possible and/or desirable. If so, the implementation and testing of the change will take place, and the new code committed back to the repository.

Submitting a change request: A description of the proposed change will be submitted to the CCPForge issue tracker as a change request to provide a record of the change process.

Evaluating a change request: The request, together with any additional information, will be evaluated by the SCM team. The request will be accepted, refined, or rejected as appropriate. The change owner will make necessary updates to the change record.

Implementation of change: Important changes to behaviour of code related to a change will be accompanied by a relevant descriptive record in the change request.

Version control: A `MAJOR.MINOR.PATCH` numerical version name scheme is used [2]. It is expected that bug fixes and relatively small changes will be accompanied by a unit increment in the patch level; more significant changes will be reflected at the minor version number; major reconstructions will occur rarely and incur a change of major version number.

10.4.4 Status

The status of changes will be tracked at the CCPForge issue tracker with a named SCM team member owning the change. The relevant tracker item will be closed when successfully implemented and tested.

The tracker is available to CPPForge member users to provide a record of changes to the code. Anonymous users will be alerted to changes via release notes.

10.4.5 Release management and delivery

There are currently no formal releases of the *Ludwig* code. Any release management and delivery will be via the SVN repository.

10.5 Plan Maintenance

The Plan is currently under review to consider organisational changes affecting the responsibilities different stakeholders.

References

[1] Adhikari, R. J.-C. Desplat, and K. Stratford, Sliding periodic boundary conditions for lattice Boltzmann and lattice kinetic equations, `arXiv:cond-mat/0503175v1` (2005).

[2] Apache Portable Runtime Project "APR's Version Numbering". See, e.g., `https://apr.apache.org/versioning.html)` (accessed November 2015).

[3] Adhikari, R., K. Stratford. M.E. Cates, and A.J. Wagner, Fluctuating Lattice Boltzmann, *Europhys. Lett.*, **71**, 473 2005.

[4] Aidun, C.K., Y. Lu, and E.-J. Ding, Direct analysis of particulate suspensions with inertia using the discrete Boltzmann equation, *J. Fluid Mech.*, **373**, 287, 1998.

[5] G.K. Batchelor *An Introduction to Fluid Mechanics*, Cambridge University Press (1967).

[6] J.R. Blake, A spherical envelope approach to ciliary propusion, *J. Fluid Mech.*, **46**, 199, 1971.

[7] M. E. Cates, J.-C. Desplat, P. Stansell, A.J. Wagner, K. Stratford, R. Adhikari, and I. Pagonabarraga, Physical and Computational Scaling Issues in Lattice Boltzmann Simulations of Binary Fluid Mixtures, *Phil. Trans. Roy. Soc. A*, **363**, 1917 (2005).

[8] B. Chun and A.J.C. Ladd, Interpolated boundary condition for lattice Boltzmann simulations in narrow gaps, *Phys. Rev. E*, **75**, 066705, 2007.

[9] Cichocki, B., and R.B. Jones, Image representation of a spherical particle near a hard wall, *Physica A*, **258**, 273, 1998.

[10] C.-H. Chang and E.I. Franses, Adsorption dynamics of surfactants at the air/water interface: a critical review of mathematical models, data, and mechanisms, *Colloids and Surfaces A*, **100** 1, 1995.

[11] Desplat, J.-C., I. Pagonabarraga, and P. Bladon, LUDWIG: A parallel lattice-Boltzmann code for complex fluids. *Comput. Phys. Comms.*, **134**, 273, 2001.

[12] H. Diamant and D. Andelman, Kinetics of surfactant adsorption at fluid/fluid interfaces: non-ionic surfactants, *Europhys. Lett.* **34**, 575 (1996).

[13] H. Diamant and D. Andelman, Kinetics of surfactant adsorption at fluid-fluid interfaces, *J. Phys. Chem.*, **100** 13732, 1996.

[14] J.-B. Fournier and P. Galatola, Modeling planar degenerate wetting and anchoring in nematic liquid crystals, *Europhys. Lett.*, **72** 403–409 (2005).

[15] J. Eastoe and J.S. Dalton, Dynamic surface tension and adsorption mechanims of surfactants at the air-water interface, *Advances in Colloid and Interface Science*, **85** 13, 2000.

[16] I. Ginzburg and D. d'Humières, Multireflection boundary conditions for lattice Boltzmann models, *Phys. Rev. E*, **68**, 066614, 2003.

[17] A. Gray and K. Stratford, TargetDP reference.

[18] Hasimoto, H., On the periodic fundamental solutions of the Stokes equation and their application to viscous flow past a cubic array of spheres. *J. Fluid Mech.*, **5**, 317.

[19] Heemels, M.W., M.H.J. Hagen, and C.P. Lowe, Simulating solid colloidal particles using the lattice-Boltzmann method, *J. Comp. Phys.*, **164**, 48, 2000.

[20] IEEE Standard 208-2005, IEEE Standard for Software Configuration Management Plans. See `http://ieeexplore.ieee.org/xpl/standards.jsp` (accessed November 2015).

[21] IEEE Standard 828-2012. IEEE Standard for Configuration Management in Systems and Software Engineering. See `http://ieeexplore.ieee.org/xpl/standards.jsp` (accessed November 2015).

[22] Jeffrey, D.J., and Y. Onishi, Calculation of the resistance and mobility functions for the two unequal rigid spheres in low-Reynolds-number flow, *J. Fluid Mech.*, **139**, 261, 1984.

[23] Kendon, V.M., M.E. Cates, I. Pagonabarraga, J.-C. Desplat, and P. Bladon, Inertial effects in three dimensional spinodal decomposition of a symmetric binary fluid mixture: A lattice Boltzmann study, *J. Fluid Mech.*, **440**, 147 (2001).

[24] Ladd, A.J.C., Numerical simulations of particulate suspensions via a discretised Boltzmann equation. Part 1. Theoretical foundation, *J. Fluid. Mech.*, **271**, 285, 1994.

[25] Ladd, A.J.C., Numerical simulations of particulate suspensions via a discretised Boltzmann equation. Part 2. Numerical results, *J. Fluid. Mech.*, **271**, 311, 1994.

[26] Ladd, A.J.C., Sedimentation of homogenous suspensions of non-Brownian spheres, *Phys. Fluids*, **9**, 491. 1996.

[27] Ladd, A.J.C., Hydrodynamic screening in sedimentating suspensions of non-Brownian spheres, *Phys. Rev. Lett.*, **76**, 1392, 1996.

[28] Ladd, A.J.C., and R. Verberg, Lattice-Boltzmann simulations of particle-fluid suspensions, *J. Stat. Phys.*, **104**, 1191, 2001.

[29] H. Li, C. Pan, and C.T. Miller, Pore-scale investigation of viscous coupling effects for two-phase flow in porous media, *Phys. Rev. E*, **72**, 026705, 2005.

[30] M.J. Lighthill, On the squirming motion of nearly spherical deformable bodies through liquid at very small Reynolds numbers, *Comm. Pure Appl. Math.*, **5**, 109, 1952.

[31] I. Llopis Fusté, *Hydrodynamic cooperativity in micro-swimmer suspensions*, Ph.D. Thesis, University of Barcelona, 2008.

[32] Message Passing Interface Forum. MPI: A Message Passing Interface Standard Version 1.3 (2008).

[33] Nguyen, N.-Q., and A.J.C. Ladd, Lubrication corrections for lattice-Boltzmann simulations of particle suspensions, *Phys. Rev. E*, **66**, 046708, 2002.

[34] T. paapnastasiou, G. Georgiou, and A. Alexandrou, *Viscous Fluid Flow*, CRC Press, Boca Raton, Florida, 2000.

[35] Paraview. See `http://www.paraview.org/`. Accessed 2011.

[36] Rapaport, D.C., *The Art of Molecular Dynamics Simulation*, Cambridge University Press, 1995.

[37] S. Succi, *The lattice Boltzmann equation and beyond*, Oxford University Press, Oxford, 2001.

[38] M. Venuroli and E.S. Boek, Two-dimensional lattice-Boltzmann simulations of single phase flow in a pseudo two-dimensional micromodel, *Physica A*, **362**, 23, 2006.

[39] R.G.M. van der Sman and S. van der Graaf, Diffuse interface model of surfactant adsorption onto flat and droplet interfaces, *Rheol. Acta* **46** 3 (2006).

[40] O. Theissen and G. Gompper, Lattice Boltzmann study of spontaneous emulsification, *Eur. Phys. J. B*, **11** 91 (1999).

[41] A.F.H. Ward and L. Tordai, *J. Chem. Phys.* **14** 453, 1946.

[42] M. Skarabot, M. Ravnik, S. Zumer, U. Tkalec, I. Poberaj, D. Babic, N. Osterman and I. Musevic, *Phys. Rev. E* **76**, 051406 (2007).

[43] D.C. Wright and N.D. Mermin, *Rev. Mod. Phys.* **61**, 385 (1989).

[44] J. Lyklema *Fundamentals of Interface and Colloid Science* Academic Press (1995).

[45] S. Mafé, J.A. Manzanares, J. Pellicer, *J.* Electroanal. Chem. **2**41, 5 7-77 (1988).

[46] F. Capuani, I. Pagonabarraga, D. Frenkel, *J.* Chem. Phys. **1**21, 973- 986 (2004).

[47] B. Rotenberg, I. Pagonabarraga, D. Frenkel, *F*arad. Discuss. **1**44, 223-243 (2010).

[48] L.D. Landau, E.M. Lifshitz, *E*lectrodynamics of Continuous Media, §15 , 2nd ed., Pergamon Press, Oxford, UK (1984).

[49] J.R. Melcher, *C*ontinuum Electromechanics, §3.10, MIT Press, Cambridge, MA, USA (1981).
downloadable from:
`http://ocw.mit.edu/ans7870/resources/melcher/resized/cem_811.pdf`

[50] L.D. Landau, E.M. Lifshitz, *T*heory of Elasticity, §3 & §16, 3rd e d., Butterworth-Heinemann, Oxford, UK (1986).