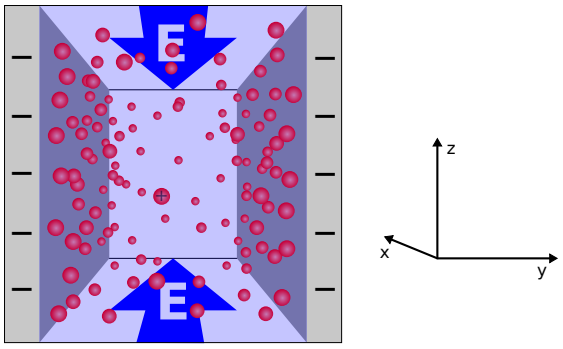
ESPResSo Tutorial

The Lattice Boltzmann Method in ESPResSo: Polymer Diffusion and Electroosmotic Flow

Stefan Kesselheim * Georg Rempfer †

October 12, 2010
Institute for Computational Physics, Stuttgart University



*kessel@icp.uni-stuttgart.de
†georg@icp.uni-stuttgart.de

Contents

1 Introduction

In this tutorial, you will learn basics about the Lattice Boltzmann Method (LBM) with special focus on the application on soft matter simulations, or more precisely on how to apply it in combination with molecular dynamics to take into account hydrodynamic solvent effects without the need to introduce thousands of solvent particles.

The LBM – its theory as well as its applications – is still a very active field of research. After almost 20 years of development there are many cases in which the LBM has proven to be fruitful, in other cases the LBM is considered promising, and in some cases it has not been of any help. We encourage you to contribute to the scientific discussion of the LBM because there is still a lot that is unknown or only vaguely known about this fascinating method.

Tutorial Outline

This tutorial has three main purposes: First, we want to make the start with the LBM as simple as possible, without leaving out the theoretical aspects of the LBM. The second purpose is to show classical examples where the LBM allows you to reproduce textbook results. The third purpose is to show how the LBM can be used with the **ESPResSo** simulation package.

Notes on the ESPResSo version you will need

The LB implementation in **ESPResSo** is currently under major revision and extension. We (the authors) hope that we could contribute to an improved usability of this feature of **ESPResSo**. Especially the boundary implementation is still very experimental and not yet published in a release version of **ESPResSo**. Currently it is only available from a still unofficial git repository. If you want to use it, please contact us.

2 The LBM in brief

Linearized Boltzmann equation

Here we want to repeat a few very basic facts about the LBM. You will find much better introductions in various books and articles, e.g. [**?** **?**]. It will however help clarifying our choice of words and we will eventually say something about the implementation in **ESPResSo**. It is very loosely written, with the goal that the reader understands how the LBM works and what **ESPResSo** does without being precise with all details.

The LBM essentially consists in solving a fully discretized version of the linearized Boltzmann equation. The Boltzmann equation describes the time evolution of the one particle distribution function, which is the probability to find a molecule in a phase space volume $\mathrm{d}x\mathrm{d}p$. In the context of LB, it is useful to apply a slightly different normalization of the one particle distribution function: The function f is normalized so that the integral over the whole phase space is the total number of particles:

$$\int f\left(x,p\right)\mathrm{d}x\mathrm{d}p = N,$$

so that the quantity $\mathrm{f}(x,p)\,\mathrm{d}x\mathrm{d}p$ corresponds to the number of particles in this particular cell of the phase space, the population.

Discretization

The LBM discretizes the Boltzmann equation not only in real space (the lattice!) and time, but also the velocity space is discretized in a surprisingly small number of velocities, in 3D typically 19, sometimes more, rarely less. Mostly we will refer to the three-dimensional model with a discrete set of 19 velocities, which is conventionally called D3Q19. These velocities are chosen so that they correspond the movement from one lattice node to another in one time step. A two step scheme is used to transport information through the system: In the streaming step the particles (in terms of populations) are transported to the cell where they corresponding velocity points to. In the collision step, the distribution functions in each cell are relaxed towards the local thermodynamic equilibrium. This will be described in more detail below.

The important hydrodynamic properties, the density, the fluid momentum density, the pressure tensor can be calculated quite straightforward from the populations: They just correspond the to

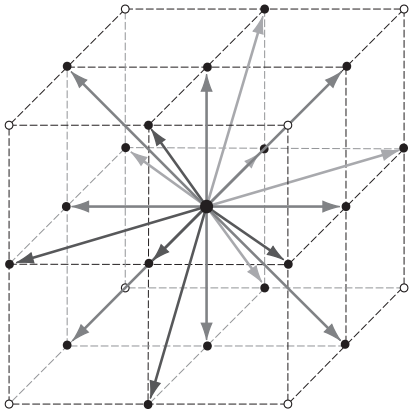


Figure 2.1: The 19 velocity vectors $\vec{c}_i$ for a D3Q19 lattice. From the central grid point, the velocity vectors point towards all 18 nearest neighbours marked by filled circles. The 19th velocity vector is the rest mode (zero velocity).

the moments of the distribution function:

$$\rho = \sum f_i \tag{2.1}$$

$$\vec{j} = \rho\vec{u} = \sum f_i \vec{c}_i \tag{2.2}$$

$$\Pi^{\alpha\beta} = \sum f_i \vec{c}_i^{\,\alpha} \vec{c}_i^{\,\beta} \tag{2.3}$$

Here the Greek indices denotes the cartesian axis and the Latin indices indicate the number in the disrete velocity set. We will occasionally refer to these quantities as the hydrodynamic fields. Note that the pressure tensor is symmetric. It is easy to see that these equations are linear transformations of the f_i and that they carry the most important information. They are 10 independent variables, but this is not enough to store the full information of 19 populations. Therefore 9 additional quantities are introduced. Together they form a different basis set of the 19-dimensional population space. This new basis set is called the modes m_i. Indeed the most practical basis is a little bit different: It contains the trace of the pressure tensor and two orthogonal vectors instead of the three diagonal elements of the pressure tensor. The 9 extra modes are referred to as kinetic modes or ghost modes. It is possible to explicitly write down the base transformation matrix, and its inverse and in the **ESPResSo** LBM implementation this basis transformation is made for every cell in every LBM step. It is possible to write a code that does not need this basis transformation, but it has been shown, that this only costs 20% of the computational time and allows for larger flexibility.

The second step: collision

The part that is now still missing is the collision part, where the actual physics happens. For the LBM it is assumed that the collision process linearly relaxes the populations to the local

equilibrium, thus that it is a linear (=matrix) operator acting on the populations in each LB cell. It should conserve the particle number and the momentum. At this point it is clear why the mode space is helpful. A 19 dimensional matrix that conserves the first 4 modes (with the eigenvalue 1) is diagonal in the first four rows and columns. Some struggling with lattice symmetries shows that four independent variables are anough to characterize the linear relaxation process so that all symmetries of the lattice are obeyed. Two of them are closely related to the shear and bulk viscosity of the fluid, and two of them do not have a direct physical equivalent. They are just called relaxation rates of the kinetic modes.

The equilibrium distribution to which the populations relax is obtained from maximizing the information entropy $\sum f_i \log f_i$ under the constraint that the density and velocity take their particular instantaneous values.

In mode space the equilibrium distribution is again very easily formulated: The modes 5-19 have the value 0 in equilibrium.

The modes are chosen so that the collision operator is diagonal:

$$\begin{aligned} m_i^\star &= \gamma_i m_i \\ \gamma_1 = \cdots = \gamma_4 &= 1 \\ \gamma_5 &= \gamma_{\mathrm{b}} \\ \gamma_6 = \cdots = \gamma_{10} &= \gamma_{\mathrm{s}} \\ \gamma_{11} &= \gamma_{\mathrm{even}} \\ \gamma_{12} = \cdots = \gamma_{19} &= \gamma_{\mathrm{odd}} \end{aligned}$$

To include hydrodynamic fluctuations of the fluid, random fluctuations are added to the modes $4 \ldots 19$ on every LB node so that the LB fluid temperature is well defined and the corresponding fluctuation formula, according to the fluctuation dissipation theorem holds. An extensive discussion of this topic is found in [1]

Particle coupling

Particles are coupled to the LB fluid with the force coupling: The fluid velocity at the point where a particle is, is calculated by a multilinear interpolation and a force is applied on the particle that is proportional to the velocity difference between particle and fluid:

$$\vec{F} = -\gamma \left(v - u \right) \tag{2.4}$$

The opposite force is distributed on the surrounding LB nodes. Additionally a random force is added to maintain a constant temperature, again according to the fluctuation dissipation theorem.

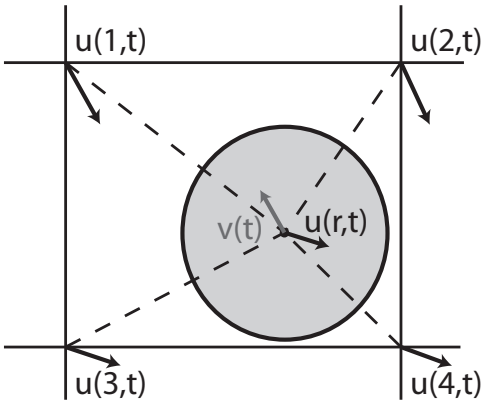


Figure 2.2: The coupling scheme between fluid and particles is based on the interpolation of the fluid velocity $\vec{u}$ from the grid nodes marking the cell the particle is in to the actual position $\vec{r}$ of the particle. This is done by linear interpolation. The difference between the actual particle velocity $\vec{v}(t)$ and the interpolated velocity $\vec{u}(\vec{r}, t)$ is used in the momentum exchange of Equation 2.4.

3 The LB interface in ESPResSo

In **ESPResSo** the LB scheme and the MD scheme are not synchronized: In one LB time step typically several MD steps are performed. This allows to speed up the simulations and is adjusted with the parameter tau. **ESPResSo** has three main commands for the LB module: **lbfluid**, **lbnode**, and **lb_boundary**. **lbfluid** is mainly used to set up parameters and does everything that concerns the whole fluid. **lbnode** involves readout and manipulation of single LB cells. **lb_boundary** allows to set boundaries, currently only the bounce back boundary method is implemented to model no-slip walls. Additionally the command **thermostat** lb is used to set the temperature.

Important Notice: All commands of the LB interface use MD units. This is convenient, as e.g. a particular viscosity can be set and the LB time step can be changed without altering the viscosity. On the other hand this is a source of a plethora of mistakes: The LBM is only reliable in a certain range of parameters (in LB units) and the unit conversion may take some of them far out of this range. So note that you always have to assure that you are not messing with that!

One brief example: a certain velocity may be 10 in MD units. If the LB time step is 0.1 in MD units, and the lattice constant is 1, then it corresponds to a velocity of 1 in LB units. This is the maximum velocity of the discrete velocity set and therefore causes numerical instabilities like negative populations.

Now the commands in more detail.

The lbfluid command

The **lbfluid** command sets global parameters of the LBM. Every parameter is given in the form **lbfluid** name value. All parameters except for gamma_odd and gamma_even are given in MD units. All parameters except for ext_force accept one scalar floating point argument.

dens	The density of the fluid.
grid	The lattice constant of the fluid. It is used to determine the number of LB nodes per direction from box_l. *They have to be compatible.*
visc	The kinematic viscosity
tau	The time step of LB. It has to be larger than the MD time step.
friction	The friction coefficient γ for the coupling scheme.
ext_force	An external force applied to every node with three components.
gamma_odd	Relaxation parameter for the odd kinetic modes.
gamma_even	Relaxation parameter for the even kinetic modes.

A good starting point for an MD time step of 0.01 is the command line

```
lbfluid grid 1.0 dens 10. visc 1. tau 0.1 friction 20.
  gamma_odd 0. gamma_even 0.
```

In combination with walls, it could also be useful to set gamma_odd and gamma_even to -0.7. E.g.

```
lbfluid grid 1.0 dens 10. visc 1. tau 0.1 friction 20.
  gamma_odd -0.7 gamma_even -0.7 ext_force 0.01 0. 0.
```

The lbnode command

The **lbnode** command allows to inspect and modify single LB nodes The general syntax is:

```
lbnode X Y Z command arguments
```

Note that the indexing in every direction starts with 0. The possible commands are:

print	Print one or several quantities to the TCL interface.
set	Set one quantity to a particular value (can be a vector)

For both commands you have to specify what quantity should be printed or modified. Print allows the following arguments:

rho	the density (scalar).
u	the fluid velocity (three floats: u_x, u_y, u_z)
pi	the fluid velocity (six floats: Π_{xx}, Π_{xy}, Π_{yy}, Π_{xz}, Π_{yz}, Π_{zz})
pi_neq	the nonequilbrium part of the pressure tensor, components as above.
pop	the 19 population (check the order from the source code please).

Example: The line

```
puts [ lbnode 0 0 0 print u ]
```

prints the fluid velocity in node 0 0 0 to the screen. The command **set** allows to change the density or fluid velocity in a single node. Setting the other quantities can easily be implemented. Example:

```
puts [ lbnode 0 0 0 set u 0.01 0. 0.]
```

The lb_boundary command

The **lb_boundary** allows to set boundary conditions for the LB fluid. In general periodic boundary conditions are applied in all directions, and only if LB boundaries are constructed finite geometries are used. This part of the LB implementation is still experimental, so please tell us about your experience with it. In general even the simple case of no-slip boundary is still an important research topic in the lb community, and in combination with point particle coupling not much experience exists. This means: Do research on that topic, play around with parameters and find out what happens.

The **lb_boundary** command is supposed to resemble exactly the constraint command of **ESPResSo**: Just replace the keyword constraint with the word **lb_boundary** and **ESPResSo** will create walls with the same shape as the corresponding constraint. Example: The commands

```
lb_boundary wall 0.5 0 0 normal 1. 0. 0.
lb_boundary wall -8.5 0 0 normal -1. 0. 0.
```

create a channel with walls parallel to the yz plane with width 8.

Currently only the so called *link bounce back* method is implemented, where the effective hydrodynamic boundary is located midway between two nodes. This is the simplest and yet a rather effective approach for boundary implementation. The **lb_boundary** command checks for every LB node if it is inside the constraint or outside and flags it as a boundary node or not. This means if the lattice constant is set to 1, the above command yields exactly the same as this:

```
lb_boundary wall 0.1 0 0 normal 1. 0. 0.
lb_boundary wall -8.1 0 0 normal -1. 0. 0.
```

This has to be kept in mind, when you use the LB boundaries.

Currently only the shapes wall, sphere and cylinder are implemented, but to implement others is straightforward. If you need them, please let us know.

4 Polymer Diffusion

In these exercises we want to use the LBM-MD-Hybrid to reproduce a classic result of polymer physics: The dependence of the diffusion coefficient of a polymer on its chain length. If no hydrodynamic interactions are present, one expects a scaling law $D \propto N^{-1}$ and if they are present, a scaling law $D \propto N^{-\nu}$ is expected. Here ν is the Flory exponent that plays a very prominent role in polymer physics. It has a value of $\sim 3/5$ in good solvent conditions in 3D. Discussions of these scaling laws can be found in polymer physics textbooks like [2–4].

We want to determine the diffusion coefficient from the mean square distance that a particle travels in the time t. For large t it should be proportional to the time and the diffusion coefficient occurs as prefactor:

$$\frac{\partial \langle r^2(t) \rangle}{\partial t} = 2dD. \tag{4.1}$$

Here d denotes the dimensionality of the system, in our case 3. This equation can be found in virtually any simulation textbook, like [5]. We will therefore set up a polymer in an LB fluid, simulate for an appropriate amount of time, calculate the mean square displacement as a function of time and obtain the diffusion coefficient from a linear fit. However we make a couple of steps in between and divide the full problem into subproblems that allow to (hopefully) fully understand the process.

4.1 Step 1: Diffusion of a single particle

Our first step is to investigate the diffusion of a single particle that is coupled to an LB fluid by the point coupling method. Investigate the script single_particle_diffusion.tcl .

In this script an LB fluid and a single particle are created and LB is used to thermostat the system. The random forces on the particle and within the LB fluid will cause the particle to move, and its position is recorded in the file pos.dat. Run the simulation script for 10000 steps and use the helper script msd.pl to calculate the MSD:

```
./msd.pl pos.dat
```

Use gnuplot to investigate the curve:

```
plot "pos.dat"
plot "msd_pos.dat"
```

What is different for short times than for long times? Can you give an explanation for the parabolic shape at short times? Use a linear fit to determine the diffusion coefficient:

```
f(x)=a*x+b
fit [1:] msd_pos.dat via a,b
```

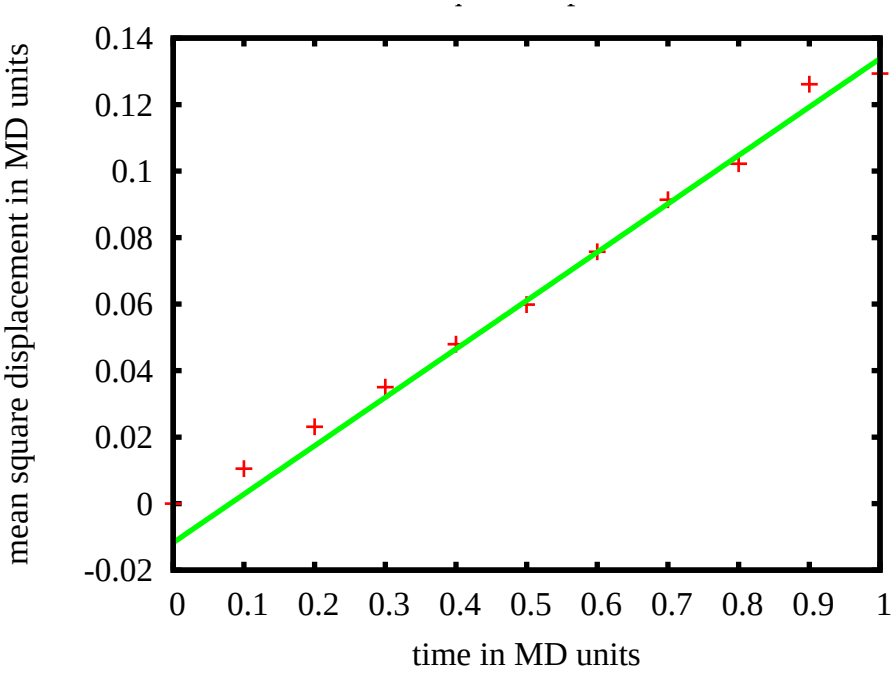


Figure 4.1: Mean square displacement of a single particle

The square brackets in the fit command tell gnuplot only to use the range right of $x = 1$ for the fit.

The file energy.dat contains the kinetic energy of the particle as a function of the elapsed simulation time. Investigate it, by plotting it with gnuplot. Calculate the average value of the kinetic energy e.g. by fitting a constant function with gnuplot. What value would you expect from a working thermostat?

Now change the box size from 16 to 8, 24, 32. What can you observe? Can you explain your observation? What happens if you replace the lb thermostat (and the LB fluid) by a Langevin thermostat?

Run the simulation again with different values for the friction coefficient, e.g. 1. 5. 20. 50. Calculate the diffusion coefficient for all cases and use gnuplot to make a plot of D as a function of γ. What do you observe? The tiny helper script fit_lin.sh (with argument msd_pos.dat) will help you with that. It contains a (quite ugly) gnuplot one-liner that does the fitting and just returns the slope. Is there any difference between the friction coefficient that you put in, and the diffusion coefficient you obain?

4.2 Step 3: The long time tail of the velocity autocorrelation function

Should we do anything here?

4.3 Step 4: Setting up a polymer

One of the typical application of **ESPResSo** is the simulation of polymer chains with a bead-spring-model. For this we need a repulsive interaction between all beads, for which one usually takes a shifted and truncated Lennard-Jones (so called Weeks-Chandler-Anderson) interaction, and additionally a bonded interaction between adjacent beads to hold the polymer together. You have already learned that the command

```
inter 0 0 lennard-jones 1. 1. 1.225 0.25 0.
```

creates a Lennard-Jones interaction with $\varepsilon = 1.$ $\sigma = 1.$ $r_{\text{cut}} = 1.225$ and εshift $= 0.25$ between particles of type 0, the desired repulsive interaction. The command

```
inter 0 FENE 7. 2.
```

creates a FENE (see **ESPResSo** manual for the details) bond interaction. Still **ESPResSo** does not know between which beads this interaction should be applied. This can be either be specified explicitly or done with the polymer command. This creates a given number of beads, links them with the given bonded interaction and places them following a certain algorithm. We will use the pruned self-avoiding walk: The monomers are set according to a pruned self-avoiding walk (in 3D) with a fixed distance between adjacent bead positions. The syntax is:

```
polymer $N_polymers $N_monomers 1.0 types 0 mode PSAW bond 0
  constraints
```

Using a random walk to create a polymer causes trouble: The random walk may cross itself (or closely approach itself) and the LJ potential is very steep. This would raise the potential energy enormously and would make the monomers shoot through the simulation box. The pruned self-avoiding walk should prevent that, but to be sure we perform some MD steps with a capped LJ potential, this means forces above a certain threshold will be set to the threshold in order to prevent the system from exploding. To see how this is done, look at the script polymer_diffusion.tcl . It contains a quite long warmup command so that also longer polymers are possible. You can probably make it shorter. Also the runtime can be reduced. You should find out about necessary number of steps by yourself.

Run the script with a polymer of chain length 16 and look at the output files which are identical to the output files of the single_particle_diffusion.tcl script. Here a Langevin thermostat is used to keep the temperature constant. Use msd.pl and fit_lin.sh to calculate the diffusion coefficient as a function of the chain length. If you are familiar with shell scripting, write a script that automatically changes the chain length.

With the help of the single particle script now add the LB fluid and calculate the MSD again. What do you observe? Do not forget to remove the Langevin thermostat after the warmup. This can be done with the command

```
thermostat off
```

Try to find out, why the results do not show the $N^{3/5}$ behaviour.

5 Electro-osmotic flow in a slit pore

Electro-osmotic flow (EOF) is the motion of water (or another liquid) induced by an electric field. It can occur e.g. in porous media, in synthetic capillaries and in vicinity of charged surfaces. Charged objects in an electrolyte solution attract ions of one species and repel ions of the other species, which gives rise to a net charge density in its neighbourhood. If an external electric field is applied, these ions are accelerated in the direction of the electric field (or oppositely if negatively charged) which causes also an acceleration of the surrounding water. In regions with zero net charge, the force on the fluid exerted by both ion species cancels, thus charged interfaces are necessary.

Conceptually electro-osmotic flow is closely related to electrophoresis, where a charged object (e.g. a polyelectrolyte) is moved by an electric field and the surrounding counterions create a flow field in the opposite direction.

In this exercise the electrokinetic equations, that allow for a classical description of the phenomenon, are introduced and you will learn how to simulate this effect with **ESPResSo** with the LBM. The special case of planar charged walls in the regime of low salt concentration can be solved analytically and you will see that we can reproduce the classical results quite well, but you will also learn about the deficiencies of both approaches. We will concentrate on the case where only one species of ions (counterions) is present. The generalization to multiple species however is straightforward.

5.1 The electrokinetic equations

We want to describe a system in which ions can diffuse under an applied field embedded in a fluid. We therefore assume that a linear convection diffusion equation is valid:

$$\vec{j} = -D\nabla c + \mu zec\vec{E} + c\vec{u} \tag{5.1}$$

Here $\vec{j}$ corresponds to the ion flux density, D corresponds to the diffusion coefficient of the ions, c to their concentration, μ to their (electrophoretic) mobility, $\vec{E}$ to the local electric field and $\vec{u}$ to the fluid velocity.

We assume that fluid fulfills the incompressible Navier-Stokes equation. The term $cze\vec{E}$ appears as source term due to the acceleration of the fluid caused by the ions. In the limit of small Reynolds numbers we can leave out the convection term and reduce to the Stokes equation.

$$\eta\Delta\vec{u} = \vec{\nabla}p + cz\vec{E} \tag{5.2}$$

The incompressibility (=continuity) equation holds:

$$\vec{\nabla}\cdot\vec{u} = 0 \tag{5.3}$$

For the electrostatic potential we make the following mean field approximation: The electric potential is caused not by single ions, but their density. This means every ion is not exposed to the instantaneous electrostatic potential but the smeared out potential of all other ions. Then the Poisson equation reads as:

$$\Delta\Phi = -c/\varepsilon \tag{5.4}$$

We will later see that this approximation is avoided in a molecular dynamics simulation of explicit ions.

This set of coupled partial differential equations is called the electrokinetic equations. In general the solution is difficult, but the planar geometry will allow us to find an analytical solution.

5.2 The slit pore geometry

We want to investigate the simplest case where EOF occurs: The flow of water through the volume between two parallel charged planes in the xy-plane. We assume that the planes are infinitely extended in the directions parallel to the plane and that the number of ions exactly cancels the charge of both planes and that the external electric field is exerted in x direction and that the position of the planes is at $x = \pm l/2$.

Some words where it comes from!!!!!

For planes with charge density σ this can be solved by:

$$\rho(y) = \frac{\varepsilon\, C^2}{2k_BT} \cdot \frac{1}{\cos^2\left(\frac{qC}{2k_BT} \cdot y\right)}, \quad \left|\frac{qC}{2k_BT} \cdot y\right| < \frac{\pi}{2}\,.$$

Here the parameter C has to fulfill the following transcendental equation:

$$C \cdot \tan\left(\frac{qd}{4k_BT} \cdot C\right) = -\frac{\sigma}{\varepsilon}, \quad 0 \le C < \frac{\pi k_BT}{2d|q|}\,.$$

Integrating the charge density twice yield the fluid flow field in the direction parallel to the applied field:

$$v_x(y) = \frac{2E\varepsilon k_BT}{\eta q} \cdot \left\{\log\left[\cos\left(\frac{qC}{2k_BT} \cdot y\right)\right] - \log\left[\cos\left(\frac{dqC}{4k_BT}\right)\right]\right\},$$
$$P(y) = P(0) + \frac{\varepsilon C^2}{2} \cdot \tan^2\left(\frac{qC}{2k_BT} \cdot y\right).$$

Here the integration constants were chosen so that no-slip boundary conditions are fulfilled a $z = \pm l/2$.

Before simulating the full system, we make two steps in between, because we need to know how to have walls in an **ESPResSo** simulation. First we want to simulate Poisseuille flow, the famous parabolic flow profile, in a slit geometry and then we want to simulation particles between two walls. Finally we combine it all to simulate the full system.

5.3 Poiseuille flow ESPResSo

Poisseuille flow is the flow through a pipe or (in our case) a slit under a homogenous force density, e.g. gravity. In the limit of small Reynolds numbers, the flow can be described with the Stokes equation. We assume the slit being infinitely extended in y and z direction and an external electric field E in y direction. No slip-boundary conditions (i.e. $\vec{u} = 0$) are located at $z = \pm l/2$. Assuming invariance in y and z direction and a steady state the Stokes equation is simplified to:

$$\eta \partial_x^2 u_y = f \tag{5.5}$$

where f denotes the force density and η the dynamic viscosity. This can be integrated twice and the integration constants are chosen so that $u_y = 0$ at $z = \pm l/2$ and we obtain:

$$u_y = \frac{f}{2\eta}\left(l^2/4 - x^2\right) \tag{5.6}$$

With that knowledge investigate the script poisseuille.tcl. Note the **lb_boundary** command. Two walls are created with normal vectors $(\pm 1, 0, 0)$. An external force is applied to every node. After 1000 LB updates the steady state should be reached.

Task: Write a loop that prints the fluid velocity at the nodes (0,0,0) to (16,0,0) and the node position to a file. Use the **lbnode** command for that. Hint: to write to a file, first open a file and then use the **puts** command to write into it. Do not forget to close the file afterwards. Example:

```
set ofile [ open "file.txt" "w" ]
puts $ofile "hello world!"
close $ofile
```

Use gnuplot to fit a parabolic profile. Can you confirm the analytic solution?

5.4 Constraints in ESPResSo

The constraint command of **ESPResSo** creates walls in the system. They have a particular "particle" type and interact with the particles present in the system with the potential defined between them. This means the distance of every particle to the constraint is calculated and used as the distance in the interaction potential.

To set up a planar channel like the LB channel before one would use the commands:

```
constraint wall dist 0.5 normal 1. 0. 0. type 1
constraint wall dist -8.5 normal -1. 0. 0. type 1
inter 0 1 lennard-jones 1. 1. 1.225 0.25 0
```

This wall is felt only by particles of type 0 and has an effective width of 6, as the potential goes steeply up at positions $x = 1.5$ and $x = 7.5$.

The syntax of the wall constraint looks weird at first, because a negative distance from the origin (first argument) is given, but the idea is that this distance times the normal vector is a point of the plane. For inclined walls this syntax is more easy to understand.

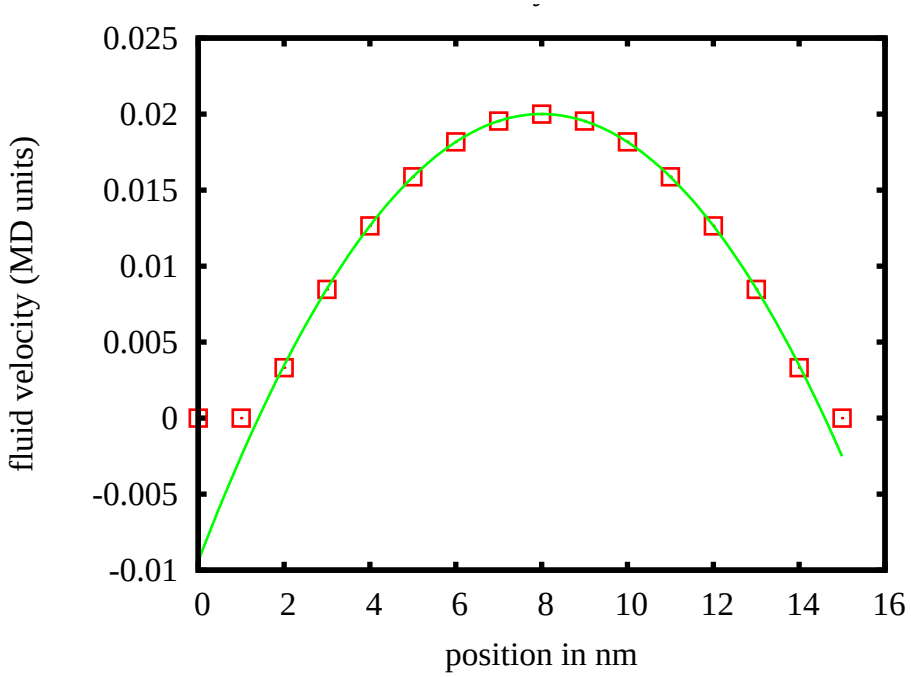


Figure 5.1: Poisseuille Flow in a slit Geometry.

ESPResSo complains every time a particle penetrates the wall, as it does not expect the particles to do so. This should normally cause no problem.

To set up a system we have, of course to make sure, that our initial configuration obeys the constraints. The easiest thing is to generate particle configurations randomly and repeat this process for every particle until a configuration is found, that is within the allowed range. Look at the script boundaries.tcl and see how that is solved. What does the script do?

5.5 Simulating EOF in ESPResSo

The last thing that is missing for the simulation of EOF is how to create a charged wall. This can be done with particles, using the fix command. The command

```
part 0 pos 1. 1. 1. q 1. fix 1 1 1
```

create a particle at the position $(1, 1, 1)$ with charge 1 that is fixed in all the spacial dimensions.

In eof.tcl two walls are created. Now use the material from the other two scripts to run the final system. Our we want to obtain a 10 Nanometer wide channel centered $x = 7.5$

1. Create constraints at $x = 1.5$ and $x = 13.5$ and create a particle-wall interaction. How can you assure that the particles creating the wall charges are not affected by interaction potential.

2. Use the particle creation method from boundaries.tcl to create particles in the system. Create a repulsive potential between them.

3. Charge the particles so that the overall system is neutral.

4. Add an electrostatic interaction with a Bjerrum length of 0.7 (the room temperature Bjerrum length of water in nanometer).

5. First do not exert an external force on the particles. Use a Langevin thermostat to bring the system to equilibrium. It will take some time for the ions to move towards the charged walls. You can use vmd to look see the process.

6. Use the density profile method from boundaries.tcl to determine the ion concentration profile. How many samples do you need to get a reasonable concentration profile.

7. Plot the concentration profile with gnuplot. Compare with the the Poisson-Boltzmann result.

8. Introduce an LB fluid with planar walls boundaries at 2.5 and 12.5. Do not forget to remove the external force if you copy&paste from poisseuille.tcl .

9. Add an external force of 0.1 to all particles that do not form the charged wall in y-direction. Now run the system for enough time steps to get a good flux and velocity profile.

10. Finally calculate the velocity profile. You will have to average over several steps. Keep in mind that number crunching with tcl is slow and that you will not need to average over too many samples.

11. Compare the flux and velocity profiles with the result from theory. Do they agree?

12. Now increase the charge density on the walls. Can you observe differences between theory and Computer simulations? They will likely be caused by the fact, that MD simulations of explicit ions automatically take into account ion correlations and the effect of the finite size of ions.

Bibliography

[1] B. Dünweg, U. Schiller, and A.J.C. Ladd. Statistical mechanics of the fluctuating lattice-boltzmann equation. *Phys. Rev. E*, 76:36704, 2007.

[2] P. G. de Gennes. *Scaling Concepts in Polymer Physics*. Cornell University Press, Ithaca, NY, 1979.

[3] M. Doi. *Introduction do Polymer Physics*. Clarendon Press, Oxford, 1996.

[4] Michael Rubinstein and Ralph H. Colby. *Polymer Physics*. Oxford University Press, Oxford, UK, 2003.

[5] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. Academic Press, San Diego, second edition, 2002.