Computing fluids: Mesoscale hydrodynamics simulations

Alberto Medina, Joscha Mecke, Marisol Ripoll

2021/01/28

Exercise for the Initial Training on Numerical Methods for Active Matter.

General instructions This exercise is an introduction to the multiparticle collision dynamics (MPC) simulation technique. It contains a simple MPC program to do two-dimensional simulation runs, in order to study the fluid behaviour in a microchannel geometry. This means that there are periodic boundary conditions in x-direction and walls in y-direction at $y = 0$ and $y = L_y$. The fluid feels a constant force proportional to the acceleration density g, "gravity", pulling it in the positive x-direction. In exercise 1, you can make yourself familiar with the method and only need to modify the input parameters in order to study the fluid behaviour. In exercise 2, you will also have to modify the source code of the simulation, compile and run. To do this exercise, you need the gcc compiler and python 2 or higher.

Useful information The MPC method is designed to simulate mesoscopic fluid dynamics. Since the fluid momentum and energy is conserved in the method, it fulfils the Navier-Stokes equation. The viscosity of the fluid comprises of two contributions. One stemming from the streaming step, η_{kin}, and one stemming from the collision step η_{col}. Thus, the full viscosity is $\eta = \eta_{\mathrm{kin}} + \eta_{\mathrm{col}}$. The two contributions can be calculated analytically. For two dimensions, the calculation yields

$$\eta_{\mathrm{kin}} = \frac{n k_{\mathrm{B}} T h}{a^2} \left\{ \frac{n/\left(1-\cos 2\alpha\right)}{n-1+e^{-n}} - \frac{1}{2} \right\} , \tag{1}$$

$$\eta_{\mathrm{col}} = \frac{m\left(1-\cos\alpha\right)\left(n-1+e^{-n}\right)}{12h} , \tag{2}$$

where h is the MPC collision time (sometimes also called dt), n is the fluid density, α is the rotation angle, a is the collision box length, m is the particle

mass and $k_{\mathrm{B}}T$ is the thermal energy. (Typically, the box length and particle mass are chosen to unity, *i.e.*, $a \equiv 1$ and $m \equiv 1$.) To derive equations (1) and (2), molecular chaos is assumed. This means that the velocities of different particles are uncorrelated. This assumption breaks down when the MPC particles tend to be in the same collision box for several time steps, because then the collisional momentum transport takes place between the same particles repeatedly.

The fluid flow $\boldsymbol{v}(\boldsymbol{x}) = (v_x(x,y), v_y(x,y))$ in the microchannel satisfies the Navier-Stokes equation, which here simplifies to

$$\partial_y^2 v_x = \frac{1}{\eta} g \,, \tag{3}$$

once the system has reached the steady-state. Hence, the resulting flowfield $\boldsymbol{v}(\boldsymbol{x})$ is constant along the x-direction and only depends on the y-direction. Integrating equation (3) and fixing the integration constants with the no-slip boundary conditions $v_x(0) = 0$ and $v_x(L_y) = 0$ yields a parabolic profile.

When you want to couple an object to the fluid, *e.g.*, a colloid or a polymer, you have to think about how to reasonably couple the dynamics of the fluid to the dynamics of the object. Here, different realisations are possible, which have different advantages and disadvantages. The most generic method is to perform each MPC step a couple of molecular dynamics steps between the object and the fluid in its vicinity. However, this method is computationally demanding and we here employ a less expensive method: We model an immobile obstacle, *e.g*, a fixed colloid, by a cluster of heavy fixed fluid particles which take part in the collision step. This acts as very high friction in the obstacle region. The internal degrees of freedom of the obstacle are updated using a molecular dynamics scheme. During the streaming step, the fluid particles may still enter the obstacle. On a microscopic level, this may be regarded as unphysical, however, on a coarse-grained level, this yields the desired behaviour.

The simulation program can be compiled as `gcc ./mpc_2d.c -o ./mpc_2d.exe`. Once you have the executable, it can be run as `./mpc_2d.exe`. The parameters are read in from the file `system_parameters.h`. You do not have to recompile after changing the parameters, but only after changing the source code. For modifying the input parameters and the source code, you can use whatever text editor you wish. If you need, do not hesitate to ask.

The simulation results are visualized with python using the plotting library matplotlib. You may find prepared plotting scripts in the folder `output/`.

Once your simulation finishes, you can plot the results by typing `./plot_flowprofile.py` or `./plot_flowfield.py`. The graphs will then be saved to `./flowprofile.pdf`, `./flowfield.pdf`, respectively. If this file already exists then it will be overwritten, so you need to rename them if you would like to save them permanently. If you change the system properties (channel size, fluid density, etc.) in the input file, you will have to edit these files as well in the python scripts so that the visualisation follows suit.

Please note the hints at the end of this document.

1. Use the files in directory `exercise1/`. It might be a good idea to copy the files to a working directory so that a pristine copy to revert to always remains.

   (a) Compile and run the simulation. Plot the flow field and the flow profile. Is the latter parabolic?

   (b) Experiment on changing the equilibration time and the total simulation time. What happens?

   (c) Measure the velocity profile for different fluid densities, time steps, and applied external forces. When do and when do not the measured and theoretical profiles agree? Why?

   (d) Switch off the grid shift by editing the grid-shift parameter in `system_parameters.h` and run the simulation again. What happens?

2. Use the code in directory `exercise2/`. It might be a good idea to copy the files to a working directory so that a pristine copy to revert to always remains.

   An unfortunate researcher attempted to optimize the code a little bit, but got something wrong. It is your task to fix it in this exercise. Each time that you change the simulation code you have to recompile.

   (a) Look at the source code, study its structure and identify where the MPC streaming and collision steps take place. The called functions `md()`, `stream()` and `collide()` can be found in the file `modules/mpc_routines/mpc_routines.h`.

   (b) The collision step is broken: The velocities in the given cell always rotate counter-clockwise. What we want is that within a given collision cell, the velocities rotate in the same direction. Whether this

is clockwise or counter-clockwise for a given cell should be random with equal probability. Fix the collision step, recompile and run the simulation and look at the flow profile.

(c) The fluid is just resting. How boring! Implement the gravity: A force, proportional to the global variable `grav` should accelerate every fluid particle in the positive x-direction. How does this change the flow profile?

(d) The friction between the fluid and the wall is not quite OK. If you look at the source code, you will see that the fluid particles bounce off the walls like billiard balls bounce off the side of the pool table. This is the slip boundary condition. What we want is the no-slip boundary condition, in which also the momentum component of the bouncing fluid particles parallel to the wall is reversed, *i.e.*, the fluid particles fly back towards the direction they were coming from. Fix the boundary condition and study the flow profile.

(e) The correct fluid-particle bouncing is not enough to produce a no-slip boundary condition. Also so-called virtual wall particles are needed. They have already been implemented but a final `#define VIRTUAL_PARTICLES` is missing in `mpc_2d.c` before the main function. Find the part of code implementing the virtual particles, and study it briefly. What does it seem to do and how does this affect the flow profile?

(f) Now everything should be fixed. Run a couple of simulations with the parameters you used in exercise 1 to check that everything works fine. To this end, compare your results with those of exercise 1.

(g) Add an obstacle to the flow by editing the respective value in `system_parameters.h` to, *e.g.*, 5. Plot the flow field and the flow profile. What happens? Try different radii.

(h) With an obstacle present, turn off the grid-shift and rerun the simulation. What happens?

Hint 1 In 2 (b), compare the two-dimensional rotation matrices which cause a rotation counter-clockwise and clockwise about an angle α.

$$\boldsymbol{R}_{\mathrm{ccw}}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{4}$$

$$\boldsymbol{R}_{\mathrm{cw}}(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \tag{5}$$

The macro `RND1` gives you a random floating point number in the range $[0.0, 1.0)$. The main function `main()` can be found in `mpc_2d.c`.

Hint 2 In 2 (c), you have to modify the streaming step. A standard integration scheme is the so-called Velocity-Verlet integration. The scheme is as follows

1. Calculate $\boldsymbol{v}(t + \frac{\delta t}{2}) = \boldsymbol{v}(t) + \frac{1}{2}\boldsymbol{a}(t)\delta t$
2. Calculate $\boldsymbol{x}(t + \delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t + \frac{\delta t}{2})\delta t$
3. Calculate $\boldsymbol{a}(t + \delta t)$ form the interactions
4. Calculate $\boldsymbol{v}(t + \delta t) = \boldsymbol{v}(t + \frac{\delta t}{2}) + \frac{1}{2}\boldsymbol{a}(t + \delta t)\delta t$.

This can be significantly simplified considering that the acceleration $\boldsymbol{a}(t)$ is constant in space and time.