3. Practical aspects of lattice-Boltzmann simulations

3.1 Introduction

After having gone through a theoretical analysis of the lattice-Boltzmann method we now will discuss what needs to be done to actually do some (computational) fluid dynamics with it.

3.2 Compressibility and the Mach number

From the analyses presented in LN02 it has become clear that the speed of sound of a lattice-Boltzmann fluid is finite which implies that we are dealing with a compressible fluid. The Chapman-Enskog analysis showed that with the LBM we are solving the Navier-Stokes equation for a compressible fluid if the bulk viscosity is two-thirds of the (shear) viscosity (Eq. 2.28). This – in general – is not the case. This issue, however, is of not much concern if we are planning to work on (near) incompressible flow problems since the bulk viscosity is of no importance if $\frac{\partial u_\gamma}{\partial x_\gamma} \approx 0$. Incompressible flow means $\mathrm{Ma} = 0$; near incompressible flow means $\mathrm{Ma} \ll 1$ with

$$\mathrm{Ma} \equiv |\mathbf{u}| / c_s \qquad (3.1)$$

the Mach number (and – as introduced before – $\mathbf{u}$ the bulk velocity and c_s the speed of sound). For incompressible flow $c_s \to \infty$. Since $c_s = \sqrt{1/3}$ (in lattice units) it means that we need to make sure that $|\mathbf{u}| \ll 1$ in lattice units. As we will see in the next section – which is about how to deal with lattice units when solving real flow systems stated in SI units – the constraint on the bulk speed $|\mathbf{u}| \ll 1$ effectively is a constraint on the time step.

3.3 From SI units to lattice units

For translating a real, physical flow problem into a LB simulation we take the example of incompressible lid-driven cavity flow. The two-dimensional geometry is shown in Figure 3.1: we have a square space filled with liquid water (density ρ=10^3 kg/m^3 (constant density, incompressible flow), dynamic viscosity μ=10^{-3} Pa·s). Of the four solid walls, the top wall is moving in the positive *x*-direction with velocity *U*. On all solid walls we want to impose a no-slip boundary conditions (liquid adjacent to a wall sticks to that wall). As a result of the motion of the top wall, the water close to the top wall will be dragged along with the top wall in the positive *x*-direction until it hits the right wall where it will be deflected in the downward direction. This creates a circulating flow in the cavity with the circulation in the clockwise direction. The cavity has a side length *L*=0.02 m (2 cm), and the top wall moves with a velocity *U*=0.01 m/s (1 cm/s). We want to simulate the flow in the cavity starting up from zero velocity and evolving to a steady state , i.e. we want to solve the bulk velocity in the cavity as a function of *x*, *y*, and *t*: $\mathbf{u}(x, y, t)$ with the lattice-Boltzmann method.

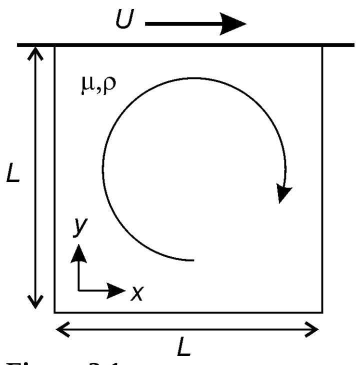

Figure 3.1

This flow problem is stated in SI units. We need to translate it into lattice units and need to make decisions regarding time step and lattice cell size. For this we translate the flow system in dimensionless form by scaling with the input variables *U* and *L* where we indicate dimensionless variables with ~:

$$\tilde{x} = x/L,\ \tilde{y} = y/L,\ \tilde{t} = tU/L,\ \tilde{\mathbf{u}} = \mathbf{u}/U \tag{3.2}$$

Dimensional analysis of the flow problem itself shows that the Reynolds number is the all-determining parameter:

$$\mathrm{Re} = \frac{\rho UL}{\mu} = \frac{UL}{\nu} \tag{3.3}$$

Based on the parameters given $\mathrm{Re} = \frac{10^3 \times 0.01 \times 0.02}{10^{-3}} = 200$. Reynolds similarity now implies that if we are able to perform an LB simulation at Re=200 with lid velocity in lattice units U_{LB} and cavity size in lattice units L_{LB}, and with the outcome of that simulation denoted as $\mathbf{u}_{LB}(x, y, t)$ (in lattice units) then the dimensionless LB result $\tilde{\mathbf{u}}_{LB} \equiv \mathbf{u}_{LB}/U_{LB}$ is the LB prediction for the actual dimensionless velocity field $\tilde{\mathbf{u}} = \mathbf{u}/U$ (as defined in Eq. 3.2) so that the LB prediction for the SI velocity field is $\tilde{\mathbf{u}}_{LB} \times U = \mathbf{u}_{LB} \times U/U_{LB}$ with *U* the lid velocity in SI units. Perhaps this is obvious but – just to make sure – I thought it is worthwhile to spell this out.

From the above we now thus are tasked with performing a lid-driven cavity LB simulation at Re=200 and we need to choose our simulation parameters. The first choice is for U_{LB}. Since we want to simulate an incompressible flow, the Mach number Ma needs to be sufficiently small. Since we expect that the maximum fluid speed in the cavity is equal to the speed of the lid, the Mach number will not exceed U_{LB}/c_s. We choose U_{LB}=0.1. With U_{LB}=0.1, U_{LB}/c_s=0.17 which is sufficiently small. The second choice is for L_{LB}. The spatial resolution of the simulation is governed by L_{LB} since L_{LB} is the number of lattice nodes in *x* and *y* direction. We now need some fluid dynamics intuition about the flow in a cavity at Re=200. This is going to be a laminar flow with not much fine-scale detail; we expect a single recirculation loop that fills the largest part of the cavity. We expect that this recirculation loop is going to be well-resolved on a 20×20 mesh so that we choose L_{LB}=20. In order to achieve

Re=200, the kinematic viscosity in lattice units thus needs to be $\nu_{LB} = \frac{U_{LB}L_{LB}}{\mathrm{Re}} = 0.01$. This can be realized by a collision operator with a relaxation time of (see Eq. 2.9) $\tau = 3\nu + \frac{1}{2} = 0.53$.

In summary: U_{LB}=0.1 motivated by limiting compressibility, L_{LB}=20 motivated by spatial resolution, $\nu_{LB} = 0.01$ for achieving the aimed for Reynolds number. If we run the simulation in lattice units, the time step is – by definition – $\Delta t = 1$. The velocity of the lid and the size of the cavity, however, determine a more physically meaningful interpretation of the time step. With U_{LB}=0.1, the lid moves over 0.1 lattice unit per time step and it thus takes $L_{LB}/U_{LB} = 200$ time steps for the lid to move once over the cavity. The Courant number of this simulation is $C = U_{LB}\,\Delta t/\Delta x = 0.1$. This is much smaller than the typically used values of $C \approx 0.8$ in explicit time stepping finite difference or finite volume solvers. The small Courant number and thus effectively the small time step are the result of the low Mach number requirement associated to the LB method.

The scaling analysis for lid-driven cavity flow is relatively easy since we were given a velocity scale in the form of the lid speed *U*. If we were to be asked to simulate the flow between two flat, vertical parallel plates as a result of gravity (see Figure 3.2) our input parameters are the distance between the plates 2*D*, the properties of the fluid (kinematic viscosity ν and the density ρ), and gravitational acceleration *g*. Dimensional analysis shows that this flow is governed by the dimensionless group $D^3 g/\nu^2$ (density does not show up since the two balancing effects of gravity and viscous forces are both proportional to the density: ρg and $\rho\nu$) so that in an LB simulation we must match this dimensionless number. Some notion of the hydrodynamics of this system will make us decide on what to choose for *D* in lattice units (if we expect laminar flow *D*=10 should be OK). We then still have two free parameters ν and *g* in lattice units to decide upon. We need to do some a priori analysis to estimate the expected flow velocities (which in this basic flow example is very simple) to come up with a ν and g combination (in lattice units) such that we match the dimensionless group and have flow speeds that are sufficiently low in order to meet the compressibility constraints.

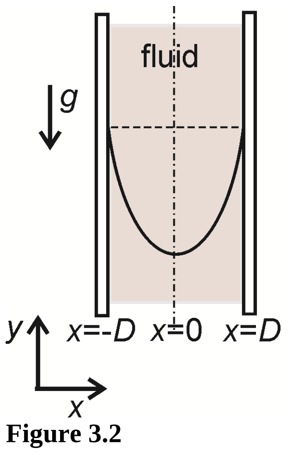

Figure 3.2

3.4 Boundary conditions

The lid-driven cavity example clearly shows the role of – and the need for – defining boundary conditions. Boundary conditions in LB simulations are a topic of active research and come in a large variety of forms and implementations. Here we discuss a few of the more basic and most used ones.

First, however, we look at one – in my mind convenient – way to implement boundary conditions. In the streaming step (Eq. 2.11) $f_i(\mathbf{x}+\mathbf{c}_i,t+1)=f_i^*(\mathbf{x},t)$, post-collision distribution functions f_i^* are transferred to neighbouring lattice sites. Lattice sites next to boundaries need to receive f_i^*'s from all directions, also from directions that (seemingly) come from outside the flow domain. This is illustrated in Figure 3.3 (left panel) that focuses on the lower-left corner of the flow domain, e.g. the lower-left corner of the lid-driven cavity. One way to implement boundary conditions is to build a layer of “ghost” cells at the other side of the boundary (the dashed cells in the right panel of Figure 3.3) and populate these ghost cells with f_i^*'s that represent the boundary condition. An LB time step would then consist of (1) collide; (2) populate ghost cells; (3) stream towards all cells *inside* the flow domain.

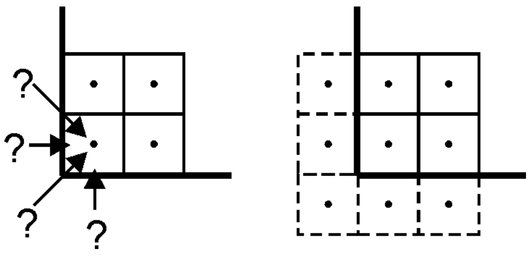

Figure 3.3

No-slip boundary condition at fixed wall
Fluid sticks at solid walls. Therefore, if the wall is not moving, the velocity at the wall is zero. A static no-slip wall can be achieved by applying a bounce-back boundary condition: the f_i's that want to leave the flow domain are bounced-back when they hit the solid wall. We show this in Figure 3.4 (left panel): the black arrows bounce on the wall and are reverted and become the red arrows. The right panel of Figure 3.4 shows how this is implemented by means of ghost cells. Figure 3.4 illustrates a so-called half-way bounce back scenario, where the actual boundary location is half a cell size away from the lattice nodes (i.e. the cell centres) that are populated with f_i's. There are also implementations – not discussed here – where nodes are located on the no-slip boundary.

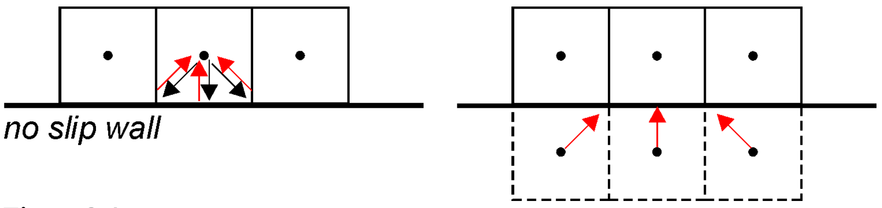

Figure 3.4

No-slip boundary condition at moving wall
A wall moving parallel to itself (as the top wall in the lid-driven cavity) requires an extension of the bounce-back condition. Consider the situation in Figure 3.5. The motion of the wall in the positive *x*-direction will add momentum to the particle that is bounced off the wall in the

positive *x*-direction, and it will reduce the momentum of the particle that is reflected in the negative *x*-direction. This is illustrated by the longer red vector that points to up-right, and the shorter red vector that points up-left. This process can be described as

$$f_5(\mathbf{x},t+1) = f_7^*(\mathbf{x},t) + 2\rho(\mathbf{x},t)u_0 w_7/c_s^2 \qquad f_6(\mathbf{x},t+1) = f_8^*(\mathbf{x},t) - 2\rho(\mathbf{x},t)u_0 w_8/c_s^2 \quad (3.4)$$

where we used the numbering of velocities as earlier given in Figure 2.1, repeated for your convenience in Figure 3.6.

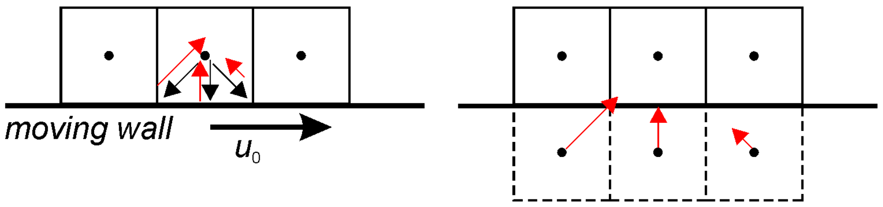

Figure 3.5

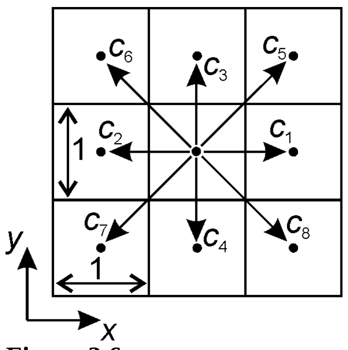

Figure 3.6

We note that change of momentum can only achieved by a change in mass. As we can see in Eq. 3.4, the net change in mass is equal to zero since $w_7 = w_8 = 1/36$.

Periodic boundary conditions
Periodic conditions imply that what leaves the domain on one side, re-enters on the other side. This is illustrated in Figure 3.7. The masses represented by the red arrows on the left of the domain (in the left panel of Figure 3.7) come from the right side of the domain. The right panel of Figure 3.7 shows how this can be implemented using ghost cells: we copy the far right layer of cells *inside* the flow domain to the ghost cells on the left *outside* the flow domain.

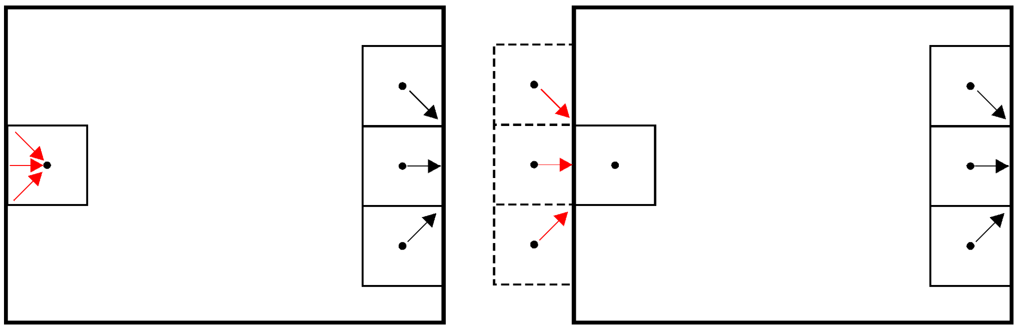

Figure 3.7

Free-slip non-penetrating boundary
A free-slip (i.e. zero shear stress) wall can be achieved by specular reflection (i.e. a mirror-like reflection) of f_i's on the boundary, see Figure 3.8. The right panel of this figure is the same as the right panel of Figure 3.4 (no-slip). Note, however, that the red arrows come from a different location: they are the reflected black arrows of the left panel.

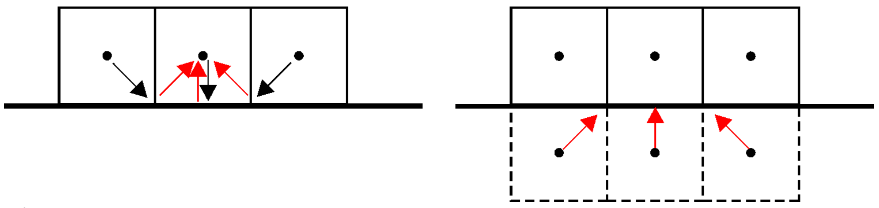
Figure 3.8

Zero-gradient boundary – an open boundary
See the example in Figure 3.9: by copying the black arrows of the left panel to the ghost cells left of the boundary (as indicated in the right panel) we mimic a no-variation in *x*-direction condition, which is a zero-gradient condition.

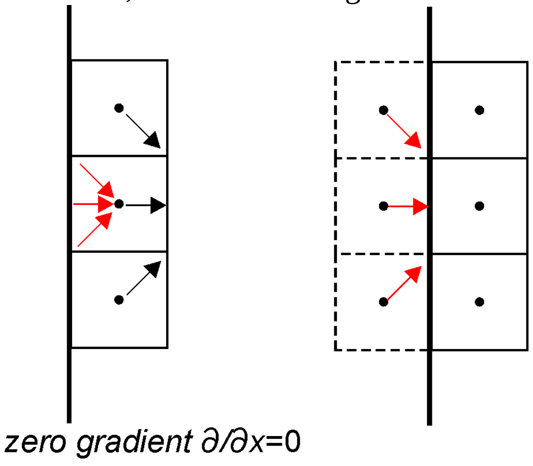

Figure 3.9

Inlet boundary
In principle, the *no-slip boundary condition at moving wall* as discussed above (see Figure 3.5) can also be applied if the velocity of the boundary has a component normal to the boundary. In that sense it can be used to impose a velocity or a velocity profile at the inlet of a flow system. I write "in principle" since I do not have experience with using this way of defining an inlet. What I usually do is applying a force on the fluid at an inlet to impose an inlet velocity, in combination with a zero-gradient boundary (see above).

Applying forces to impose boundary condition relates to immersed boundary methods.

Immersed boundaries
Immersed boundary methods are applied over a wide spectrum of approaches to simulating fluid flow, the LB method being no exception. The idea is to locally exert forces on the flow such that as a result of these forces the velocity at certain locations in the flow takes on desired values (e.g. zero velocity if we want to impose a static no-slip condition).

A significant advantage of the immersed boundary method is the possibility to impose off-grid conditions and/or moving boundaries. When applying immersed boundaries for off-grid (e.g. curved) boundaries, one uses interpolation to estimate velocities at points that do not coincide with lattice nodes. One uses extrapolation to distribute the forces acting on the fluid at points not coinciding with lattice nodes to lattice nodes. For better appreciating the workings of the immersed boundary method we first need to know how to apply forces to an LB fluid. This topic will be discussed in LN04 where we then also will revisit the immersed boundary method.

3.5 Coding

The ingredients of an LB computer code are
(1) Definitions: set grid size $nx \times ny$, number of time steps, kinematic viscosity ν, weighing factors w_i.
(2) Initialize $f_i^*(\mathbf{x}, t=0)$, e.g. by setting it equal to the equilibrium distribution function with uniform ρ and $\mathbf{u} = \mathbf{0}$.
Start the time stepping loop; what comes now is done each time step
(3) Fill the ghost cells with boundary values of f_i^*
(4) Stream
(5) Collide & return to (3) until the pre-set number of time steps is completed
Leave the time stepping loop and write to file, plot,… etc.

A sample code in Matlab (lusby2.m named after a 4th year project student) for lid-driven cavity flow has been distributed under the course participants. It is a D2Q9 code with velocity numbering as defined in LN02.

Below are coding suggestions regarding some of the main ingredients of the LB algorithm. The pieces of computer code are in Matlab with the exception of the array (or vector if you wish) indices. Matlab does not allow these to be smaller than 1. Since our velocity direction numbering starts with $i = 0$, I decided – for clarity – to keep this numbering in the code fragments below.

Streaming
In streaming we execute Eq. 2.11 (repeated here as Eq. 3.5):

$$f_i(\mathbf{x} + \mathbf{c}_i, t+1) = f_i^*(\mathbf{x}, t) \qquad (3.5)$$

After we have dealt with our boundary conditions via populating ghost cells (as explained in the previous section) this can be implemented in computer code as a loop over all the active (i.e. non-ghost) lattice nodes. One could be tempted to work with two arrays `f(0…8,1…nx,1…ny)` for the left-hand side of Eq. 3.5 and `fstar(0…8,0…nx+1,0…ny+1)` for the right-hand side and with `nx` the number of active lattice nodes in *x*-direction and `ny` the number of active nodes in *y*-direction. Note that `fstar` needs to include ghost cells, hence the extension to `0…nx+1,0…ny+1` where `f` has `1…nx,1…ny`. Then do something like this piece of *pseudo* code (where I refer to the velocity numbering in Figure 3.6).

```
for j=1:ny
  for i=1:nx
```

```
    f(0,i,j)=fstar(0,i,j)
    f(1,i,j)=fstar(1,i-1,j)
    f(2,i,j)=fstar(2,i+1,j)
    f(3,i,j)=fstar(3,i,j-1)
    f(4,i,j)=fstar(4,i,j+1)
    f(5,i,j)=fstar(5,i-1,j-1)
    f(6,i,j)=fstar(6,i+1,j-1)
    f(7,i,j)=fstar(7,i+1,j+1)
    f(8,i,j)=fstar(8,i-1,j+1)
  end
end
```

This is not a smart way to do streaming as it requires two large arrays (specifically in 3D and for big computations these arrays can get huge) and thus eats up lots of computer memory. We can do, however, with one array `f(0…8,0…nx+1,0…ny+1)`. At the start of the streaming operation this array contains f_i^*, including its values in the ghost nodes. Then we run the following

```
for j=1:ny
  for i=1:nx
    f(2,i,j)=f(2,i+1,j)
    f(4,i,j)=f(4,i,j+1)
    f(7,i,j)=f(7,i+1,j+1)
    f(8,i,j)=f(8,i-1,j+1)
  end
end
for j=ny:-1:1
  for i=nx:-1:1
    f(1,i,j)=f(1,i-1,j)
    f(3,i,j)=f(3,i,j-1)
    f(5,i,j)=f(5,i-1,j-1)
    f(6,i,j)=f(6,i+1,j-1)
  end
end
```

Note that in the second double loop we go in reverse order (the `-1` in the `for` statements). The trick here is to update from f_i^* to f_i in the same direction as you go through the loops so that on the right side of the equal sign there actually is an f_i^* and not an already updated f_i. In one dimension this is relatively simple. In two dimensions (as is the case here) one should realize that the data is ordered as

$$(i,j):\quad \left[(1,1)(2,1)\cdots(nx,1)\right]\left[(1,2)(2,2)\cdots(nx,2)\right]\cdots\cdots\left[(1,ny)(2,ny)\cdots(nx,ny)\right]$$

Filling in the ghost nodes

As an example of filling the ghost nodes with the relevant f_i^* values, here is how it can be done for the bottom wall of the cavity, which is a no-slip wall. The first row of active nodes above the bottom has `j=1.` Therefore, the row of ghost nodes underneath the bottom wall has `j=0`. The velocity directions entering through the bottom are 3, 5, 6 (see Figure 3.6). The opposite velocity directions are 4, 7, 8 respectively. The associated f_i^*'s come from one layer above, hence the `j+1` on the right-hand side. As can be seen in Fig. 3.4, the f_i^*'s of the diagonal velocity directions need to be shifted in the *x*-direction, hence the `i+1` and `i-1` on the right hand size for directions 5 and 6.

```
% bounce back at bottom
j=0;
for i=1:nx
  x(3,i,j)=x(4,i,j+1);
  x(5,i,j)=x(7,i+1,j+1);
  x(6,i,j)=x(8,i-1,j+1);
end
```

Note that in the above we use (as in lusby2.m) `x` instead of `f`.

Calculating the equilibrium distribution & collide
Here is a piece of code dealing with how to do the collision step (Eq. 2.10, LN02).
We loop over all active nodes.
We do a few things to try and speed up the computations (where it should be noted that everything that is calculated inside the loop is calculated for each node and each time step so that computational efficiency inside the loop pays off):

write out the expression for `rho` instead of putting it in a loop
multiplication is cheaper than division, we do one division to determine `1.0/rho` and then use `rrho` twice
determine `usq=ux*ux+uy*uy` and using it multiple times
similar for `cdotu`

```
for j=1:ny
  for i=1:nx
    rho=x(0,i,j)+x(1,i,j)+x(2,i,j)+x(3,i,j)+x(4,i,j)+x(5,i,j)+x(6,i,j)+
        x(7,i,j)+x(8,i,j);
    rrho=1.0/rho;
    ux=x(1,i,j)-x(3,i,j)+x(5,i,j)-x(6,i,j)-x(7,i,j)+x(8,i,j);
    ux=ux*rrho;
    uy=x(2,i,j)-x(4,i,j)+x(5,i,j)+x(6,i,j)-x(7,i,j)-x(8,i,j);
    uy=uy*rrho;
    usq=ux*ux+uy*uy;
    xeq(0)=m0*rho*(1.0-1.5*usq);
    cdotu=ux;
    xeq(1)=m1*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    cdotu=uy;
    xeq(2)=m1*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    cdotu=-ux;
    xeq(3)=m1*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    cdotu=-uy;
    xeq(4)=m1*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    cdotu=ux+uy;
    xeq(5)=m2*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    cdotu=-ux+uy;
    xeq(6)=m2*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    cdotu=-ux-uy;
    xeq(7)=m2*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    cdotu=ux-uy;
    xeq(8)=m2*rho*(1.0+3.0*cdotu+4.5*cdotu*cdotu-1.5*usq);
    for l=0:8
      x(l,i,j)=(1.0-rtau)*x(l,i,j)+rtau*xeq(l);
    end
  end
end
```

Open-source lattice-Boltzmann codes

If you do not feel like writing your own code, there are open-source code available online, e.g.
http://www.openlb.net/
http://www.palabos.org/