2 Lattice Gas Models

Lattice gas cellular automaton models were the harbingers of LBM. We dedicate a chapter to basic cellular automata and lattice gases in part out of historical interest and in part because they represent a somewhat simpler and possibly more intuitive framework for learning gases on a lattice. Unfortunately, they require a perhaps less familiar Boolean mathematics (base 2 integers) on a less familiar triangular lattice. This material is not essential to applying LBM but it is interesting in its own right and might be helpful to developing a fuller understanding of LBM.

2.1 Cellular Automata

A cellular automaton (CA) is an algorithmic entity that occupies a position on a grid or lattice point in space and interacts with its identical neighbors. A cellular automaton generally examines its own state and the states of some number of its neighbors at any particular time step and then resets its own state for the next time step according to simple rules. Hence, the rules and the initial and boundary conditions imposed on the group of cellular automata uniquely determine their evolution in time.

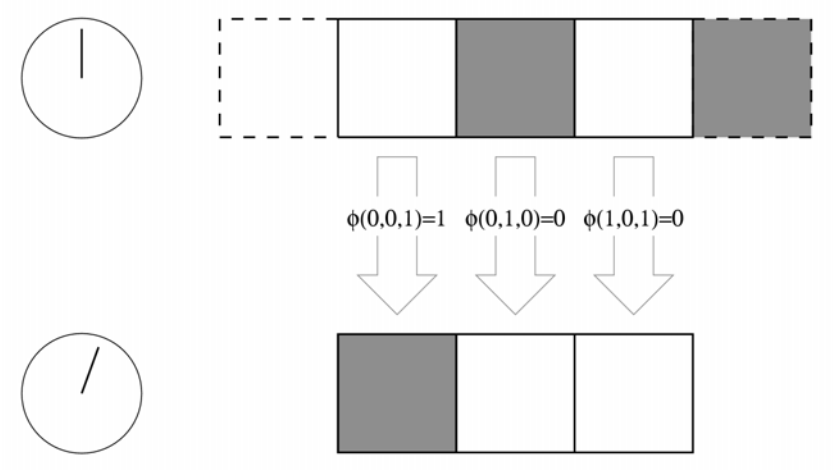

Figure 8. The basic components of a cellular automaton: a tiling of space, a clock that ticks out time, and a transition or update rule. The tiling of space in this illustration is a row of cells in a 1D space. A clock is represented at the left of the tilings. The update rule is denoted by the arrows from the state of the tiling from one time to the next. The update rule in this illustration maps the on/off state of a cell and its two neighbors at a given time tick on the clock to the on/off state of the cell at the next time tick on the clock.

The simplest cellular automata models are those that exist in one dimension on a line and consider only their own states and those of their two nearest, adjacent neighbors. If these automata have only two possible states (0 and 1, for example), then there are 256 possible rules for updating the central automaton. We can write the update rule symbolically as $a_i' = \phi(a_{i-1}, a_i, a_{i+1})$ where a_i' is the updated state, ϕ is one of 256 functions, and a_i, a_{i-1}, and a_{i+1} are the initial states of the automaton itself and its left and right neighbors respectively.

Many cellular automata can be computed using binary arithmetic. Wolfram (1986, 2002) presented a complete classification and analysis of the 265 rules for the 2-state, 2-neighbor automata. For each binary number from 00000000 to 11111111 (decimal 0 to 255) the update function ϕ is defined as follows.

Proceeding from right to left, each binary digit represents 2^0, 2^1, 2^2, … , 2^7. Hence, the binary number 00000001 is $2^0 = 1$ while 00010010 is $2^4 + 2^1 = 18$. We take the exponents (4 and 1 in the case of binary 00010010)

as a variable n and solve for n_2, n_1, and n_0 in $n = 4n_2 + 2n_1 + n_0$. Then $\phi(n_2, n_1, n_0) = 0$ or 1 depending on the value of the binary digit. This is best illustrated by an example. The rightmost binary digit in 00010010 is 0. Its exponent in 2^0 is also 0. Hence, $n = 0$ and the only solution of $n = 4n_2 + 2n_1 + n_0$ is $n = 0 = 4(0) + 2(0) + (0)$ or $n_2 = n_1 = n_0 = 0$. Finally, $\phi(0, 0, 0) = 0$. Therefore, if the central automaton and its two nearest neighbors all have state 0 at a time step, the central automaton will be in the 0 state at the next time step.

The second binary digit is 1 and its exponent $n = 1 = 4(0) + 2(0) + (1)$. Thus, $\phi(0, 0, 1) = 1$. So, if the right neighbor has state 1 and the left neighbor and the central automaton are in the 0 state at a time step, the central automaton will update to state 1 at the subsequent time step.

The third binary digit is 0 and its exponent $n = 2 = 4(0) + 2(1) + (0)$. Thus, $\phi(0, 1, 0) = 0$ and, if the central automaton has state 1 and the right and left neighbors are in the 0 state at a time step, the central automaton will update to state 0 at the subsequent time step.

If we complete these computations for every combination of the 2 states for each of the 3 automata, we arrive at the following update table that defines ϕ:

$\phi(0, 0, 0) = 0$
$\phi(0, 0, 1) = 1$
$\phi(0, 1, 0) = 0$
$\phi(0, 1, 1) = 0$
$\phi(1, 0, 0) = 1$
$\phi(1, 0, 1) = 0$
$\phi(1, 1, 0) = 0$
$\phi(1, 1, 1) = 0$

In general, for n_s states and a neighborhood of n_n automata (including the one to be updated), the update table will require $n_s^{n_n}$ entries.

Despite the simplicity of this cellular automaton, it displays a complex evolution classed as chaotic and aperiodic by Wolfram (1986). To visualize its behavior, we can begin with a random initial condition of states, apply the update table, and show subsequent generations as a sequence of lines (Figure 9). This can easily be implemented on a spreadsheet. The ex-

ercises at the end of the chapter provide some hints. These and far more elaborate CA are discussed in Wolfram (2002).

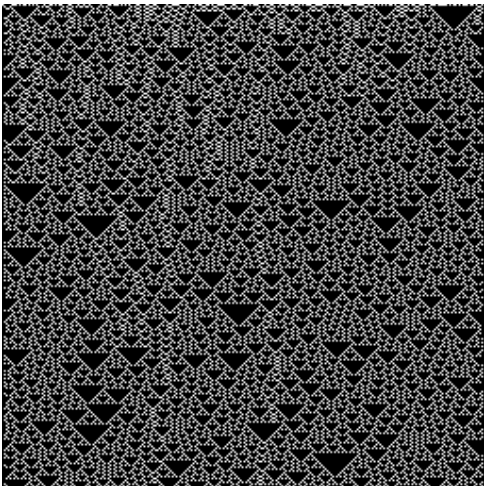

Figure 9. Evolution of a 1-dimensional, 2-state, 2-neighbor cellular automaton. Initial condition has 50% probability of sites in state 1 (black, top line). Subsequent generations are shown in each line progressing from top to bottom.

2.2 Two-Dimensional Lattice Gas Model of Fluid Flow

Lattice gas cellular automata were presented as a viable means of solving the Navier-Stokes equations of fluid motion in a landmark paper that appeared in 1986. Frish, Hasslacher, and Pomeau (Frish et al. 1986) provided the first lattice gas model that could properly simulate the 2-dimensional Navier-Stokes equations. It is commonly referred to as the 'FHP' model after these authors. This model is constructed on a equilateral triangular lattice that provides an isotropic solution. Lattice points are separated by 1 lattice unit (*lu*) and all particles have only one speed: 1 lu/time step ($lu\ ts^{-1}$). At each lattice point **x**, there may be up to 6 particles – one for each of the possible velocities defined by the particle speed and one of the six possible directions: $\mathbf{e}_a = (\cos \pi a/3, \sin \pi a/3)$ where $a = 1, 2, \ldots, 6$, and $\mathbf{e}_a$ is the velocity vector pointing from the origin (0,0) to the Cartesian coordinate ($\cos \pi a/3, \sin \pi a/3$). A string of Boolean variables $\mathbf{n} = (n_1, n_2, \ldots, n_6)$ contains the states ($n_a = 0$ or 1) indicating the presence (1) or absence (0) of

particles moving from a lattice gas site at $\mathbf{x}$ to a neighboring site at $\mathbf{x} + \mathbf{e}_a$ (Rothman and Zaleski 1997).

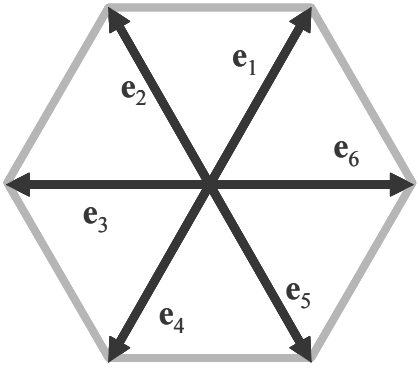

Figure 10. FHP unit velocity vectors.

The evolution of a lattice gas model proceeds in two steps that take place during each time step. The first step is a propagation, 'hopping' or ‘streaming’ step in which the particles move to new sites according to their previous positions and their velocities. Next, the particles collide and scatter according to collision rules.

2.2.1 Collision Rules

There are several possible types of collisions on the hexagonal lattice. Only two types are considered in the simplest FHP model; two-body collisions involve 2 particles while three-body collisions involve 3. Two critical features of the lattice gas that allow it to simulate the Navier-Stokes equations are mass conservation and momentum conservation. Thus, it is essential that the microscopic-scale collisions honor mass and momentum conservation. In the lattice gas, all particles have the same mass and speed so that momentum conservation reduces to conservation of the vector sum of the velocities. Head-on collisions between 2 particles (or 3 particles approaching one another from $\pi/3 = 120°$ separation) have no net momentum. Hence, the results of these collisions must also have zero net momentum.

Figure 11 illustrates the zero net momentum, 2- and 3-particle collisions respectively. The vectors shown in these figures represent velocity vectors

attributable to particles at the center of each hexagon just prior to and just after the collision step.

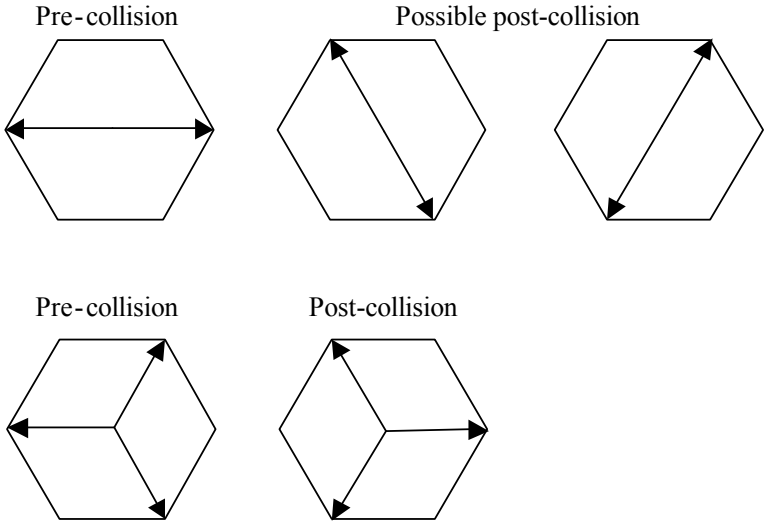

Figure 11. Zero net momentum, head-on, 2- and 3-particle collisions.

2.2.2 Implementation

In practice, the directions are coded to a variable A though F as shown in Figure 12.

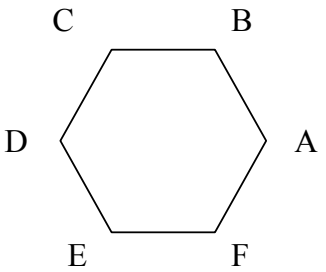

Figure 12. FHP Variable definitions

The variables correspond to bit strings as shown in Table 1. Note that only the first six bits (through a value of 32) are needed to describe the presence or absence of particles in all six directions. The additional bits play important roles however. The seventh bit signals the presence of a solid, while the eight bit is randomly 0 or 1 simply to decide between alternative post-collision states like those at the top of Figure 11.

Table 1. FHP model variables and their values.

	Bit Value							
	128	64	32	16	8	4	2	1
A	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	1	0
C	0	0	0	0	0	1	0	0
D	0	0	0	0	1	0	0	0
E	0	0	0	1	0	0	0	0
F	0	0	1	0	0	0	0	0
S	0	1	0	0	0	0	0	0
R	1	0	0	0	0	0	0	0

Now consider the collision illustrated in Figure 13. There is no change in the pre- and post-collision velocities because no other velocity configuration that is possible on the hexagonal lattice conserves the momentum present prior to the collision. The same is true for 2 particles that collide at 120º and all three-particle collisions with the exception of that shown in Figure 11.

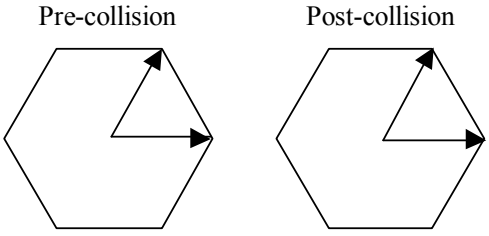

Figure 13. Pre- and post-collision velocities for 2-particle collision with initial velocities separated by 60°. No configurations other than the original conserve mass and momentum; the same is true for all 5- and 6-particle collisions.

Inclusion of these 2- and 3-particle collisions completes the simplest lattice gas model. More complex models that include 4-particle collisions and particles with zero velocity can be devised (Rothman and Zaleski, 1997). Five- and 6-particle collisions cannot be replaced with any other velocity configuration if momentum is to be conserved.

With these considerations, we are in a position to construct a look-up table (Figure 14) that reads the current configuration of particle velocities, solid presence, and random bit, and returns the new configuration. We begin by filling the table with the trivial information 'new configuration = old configuration' for each of the 256 possible configurations, because in fact we have decided many configurations will not change. For the first 64 configurations (00000000 through 00111111) the seventh bit (which signifies a solid surface) is 0 and no solid is present. The same is true for configurations 128 through 191 (10000000 through 10111111). We continue to assume (for the moment) that no changes to these configurations will be needed.

	Bit Value							
	128 R	64 S	32 F	16 E	8 D	4 C	2 B	1 A
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
…								
255	1	1	1	1	1	1	1	1

Figure 14. Collision “look up” table. 256 entries. Start with all unchanged. ‘new configuration’ = ‘old configuration’

In contrast, for configurations 64 through 127 and 192 through 255 (01000000 through 01111111 and 11000000 through 11111111), there are solids present. The particles cannot pass through the solids. One of the simplest boundary conditions to apply at the surface of the solid is the 'bounce back' condition. This consists of sending the particle directly back where it came from. So, for instance (referring to Figure 12), an A particle becomes a D particle, B becomes E, C becomes F, and so on. These are the first modifications we make to the table (see Figure 15).

	In-State Bit Value							
	128 R	64 S	32 F	16 E	8 D	4 C	2 B	1 A
64	0	1	0	0	0	0	0	0
65	0	1	0	0	0	0	0	1
66	0	1	0	0	0	0	1	0
…								
127	0	1	1	1	1	1	1	1

	Out-State Bit Value							
	128 R	64 S	32 F	16 E	8 D	4 C	2 B	1 A
64	0	1	0	0	0	0	0	0
72	0	1	0	0	1	0	0	0
80	0	1	0	1	0	0	0	0
…								
127	0	1	1	1	1	1	1	1

Figure 15. In-State/Out-State bit values with solids present (S = 1).

Next we take into account the collisions considered in Figure 11. There are three obvious possibilities for the two-particle, head on collision: AD, BE, and CF. Actually however, because of the eight bit and because there are two possible outcomes that must be equally likely, there are really two sets of these; one set has the eight bit = 0 and the other has it equal to 1. Now, for example, say we have the configuration AD with bit 8 = 0 (00001001 or 9). The new table configuration is BE (00010010 or 18). If the eight bit is 1, AD is 10001001 (= 137). Now the other configuration – CF (10100100 = 164) is selected. Bit 8 remains unchanged.

The randomness introduced by this procedure is essential to the ability of the lattice gas to simulate fluids.

	In-State Bit Value							
	128 R	64 S	32 F	16 E	8 D	4 C	2 B	1 A
9 (AD)	0	0	0	0	1	0	0	1
137 (AD)	1	0	0	0	1	0	0	1

	Out-State Bit Value							
	128 R	64 S	32 F	16 E	8 D	4 C	2 B	1 A
18 (BE)	0	0	0	1	0	0	1	0
164 (CF)	1	0	1	0	0	1	0	0

Figure 16. In-State/Out-State Bit values for two-particle head-on collisions.

Changing all of the zero-momentum, 3-particle collisions (ACE, BDF, and their bit 8 = 1 counterparts (Figure 17)), completes the definition of the model. Each of the 256 unique combinations of the 8 bits is accounted for.

	In-State Bit Value							
	128 R	64 S	32 F	16 E	8 D	4 C	2 B	1 A
21 (ACE)	0	0	0	1	0	1	0	1
42 (BDF)	0	0	1	0	1	0	1	0
149 (ACE)	1	0	0	1	0	1	0	1
170 (BDF)	1	0	1	0	1	0	1	0

	Out-State Bit Value							
	128 R	64 S	32 F	16 E	8 D	4 C	2 B	1 A
42 (BDF)	0	0	1	0	1	0	1	0
21 (ACE)	0	0	0	1	0	1	0	1
170 (BDF)	1	0	1	0	1	0	1	0
149 (ACE)	1	0	0	1	0	1	0	1

Figure 17. In-State/Out-State Bit Values for three-particle head-on collisions.

Because there are relatively few in-state => out-state changes in our table, it is easy to implement this in computer code. The following is from code provided by Rothman and Zaleski (1997). A ‘table’ array is indexed with the possible in-states from 0 to 255 and the out-states are the values contained in the array.

```
table[A + D]       = B + E;
table[B + E]       = C + F;
table[C + F]       = A + D;
table[A + D + EPS] = C + F;
table[B + E + EPS] = A + D;
table[C + F + EPS] = B + E;

table[A + C + E]       = B + D + F ;
table[B + D + F]       = A + C + E;
table[A + C + E + EPS] = B + D + F + EPS;
table[B + D + F + EPS] = A + C + E + EPS;
```

Figure 18. Code fragment from Lgapack Version 0.98 for the simulation of flow with lattice gas automata. Copyright (C) 1997 D.H. Rothman and S. Zaleski. This code is freely available under GNU General Public License from ftp://ftp.jussieu.fr/jussieu/labos/lmm/Lgapack/. EPS refers to the random bit R.

Three additional details are needed to implement a FHP model. First, the need to select randomly among the two possible configurations for the head-on two-particle collisions interjects a great deal of ‘noise’ into the simulations. In fact, this ‘noise’ is crucial to the ability to simulate hydrodynamics with a lattice gas and may even be viewed as an advantage in certain circumstances. But to obtain the smooth flow fields we expect in fluids at macroscopic scales under many conditions from a lattice gas simulation, a significant amount of temporal and/or spatial averaging is needed. Next, the equilateral triangular lattice is not particularly amenable

to computer computation (instead of the four or eight neighbors of a Cartesian point, there are six) and a remapping scheme is needed (Figure 19). The vertical separation of node points is $\sqrt{3}/2$.

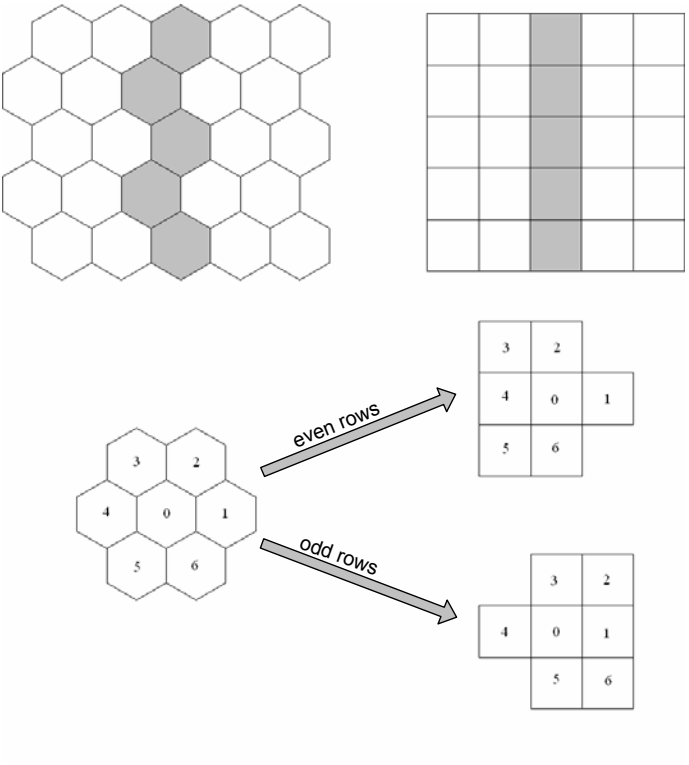

Figure 19. One possible scheme for remapping the equilateral triangular grid onto a more 'computer-friendly' system. Alternate rows are shifted left or right in the scheme.

Finally, a driving force is needed. The simplest approach is to 'flip' the momentum of some randomly selected fraction of the particles; for example, a fraction of the D direction particles become A direction. This corresponds to the addition of A-direction momentum to the system.

2.2.3 Example

With these pieces in place, we can compute reasonable hydrodynamics. Figure 20 shows the results of an early lattice gas simulation (see Rothman (1988) for a similar simulation). As noted by others (Rothman and Zaleski 1997; Succi 2001; Wolfram 2002), the model is remarkable for its great simplicity. That fluid flows can be computed on the basis of only 6 particle momenta and a handful of collisions attests to an amazing underlying simplicity in nature.

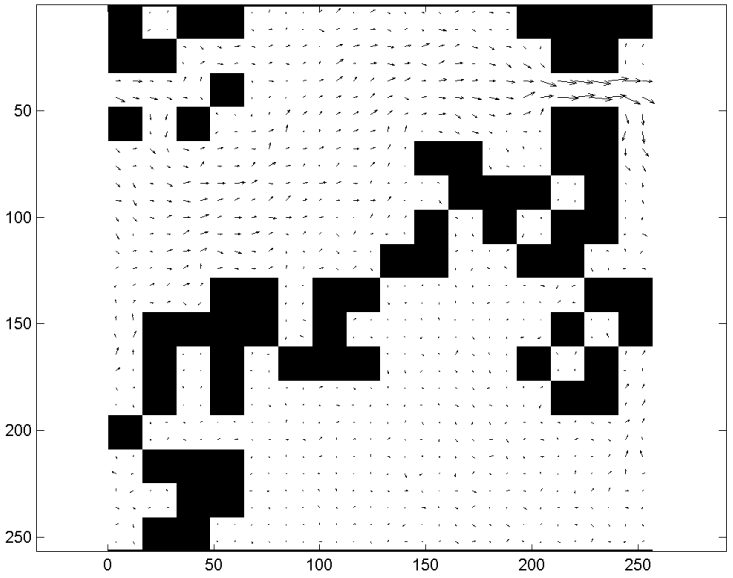

Figure 20. Crude lattice gas simulation of flow in a periodic 2-D network.

2.3 Exercises

1. Implement Wolfram's (1986, 2002) Rule 18 CA on a spreadsheet. In Microsoft Excel®, the 2nd row, 2nd column formula is:

```
=IF(OR(AND((A1=1),(B1=0),(C1=0)),AND((A1=0),(B1=0),(C1=1))),1,0).
```

It needs to be copied throughout the domain except on row 1, which is the initial condition. Try different initial conditions by seeding the first line with different patterns of 0s and 1s.

2. Download LGAPACK from the ftp site ftp://ftp.jussieu.fr/jussieu/labos/lmm/Lgapack/. Read the README file and study the code fhp6_simp4.c and the associated header (.h) files. Delete the original fhp6_simp.c and rename fhp6_simp4.c to fhp6_simp.c, then type 'make clean' and 'make' to compile the code. Modify the parameters.h file so that FORCING_RATE = 0.0001 and TPRINT, TMAX, and TAVG all equal 100000. Run the code and use the following MATLAB® code to visualize the results:

```
clear('all')
load x_mom
load y_mom
load mass
x_vel=x_mom./(2*mass)
y_vel=sqrt(3)*y_mom./(2*mass)
quiver(x_vel',y_vel')
axis equal
```

You should obtain a Poiseuille velocity profile. Reduce TPRINT, TMAX, and TAVG to 1000. What is the effect on the results and why?

3. Find the maximum x velocity in the simulation and use the velocity, FORCING_RATE (g), and channel width in Eq. (4) to estimate the kinematic viscosity of the simulated fluid. (Note that the densities in G^* and the bulk viscosity cancel, leaving g and the kinematic viscosity.) Compare your estimate to the theoretical density-dependent kinematic viscosity of the lattice gas model value given by (Rothman and Zaleski, 1997)

$$\nu = \frac{1}{12 f (1 - f)^3} - \frac{1}{8}, \tag{8}$$

where f is the ‘reduced density’ (average number of particles per lattice link) as given in the parameters.h file. Can you improve your estimates by changing the simulation?