Simple Discrete Simulations

Software project May 2013

1 Overall Plan

We start with a toy system to make sure we can write elegant, bug-free programs. Over the coming weeks, we make the system and program more complicated. This hand-out gives an overview. The extensions will be added in coming hand-outs.

2 Introduction

It is most elegant to work out the properties of a system analytically. Sometimes it is easier to simulate and see what happens. For example, you might say diffusion is well described by Fick's equation, but could you describe the system analytically if we add walls or obstacles ? If you have two charged particles, an equation for their interactions and initial velocities, you can describe their paths (trajectories) analytically. If you have 10 particles, there is no known analytical description. Here, we consider a problem where there is an analytical form, then consider a slight complication which makes it much harder to simulate.

Particle simulations are common in physics and chemistry when you have a model for the interactions between particles. In astronomy, the main interaction is gravity and you might simulate the paths of planets. In chemistry, the interactions the interactions are different and you may not be interested in the individual particles and their exact paths. You are probably interested in average behaviour, so you can calculate properties such as density, viscosity, heat transport or compressibility.

We will describe particles in boxes, with simple interactions. This project will begin very simply, but become more complicated. We will start with the distribution of particles in boxes and use this to establish the basic machinery. If all goes well, we will try to reach ion-selective membranes.

3 Models

All of the systems will be two dimensional. In the *x* direction, we will look at motions. In the *y* direction, we will have hard walls.

3.1 Particles

We will consider three models.

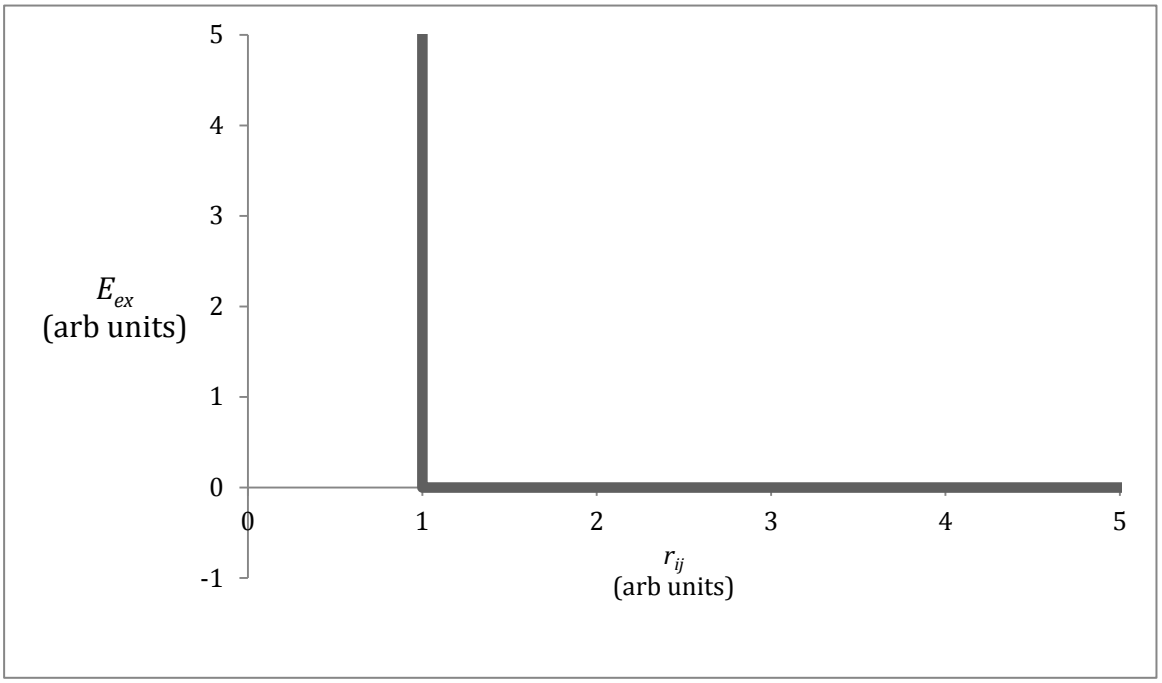
3.1.1 Excluded volume

In many polymer simulations, there is very little attraction between particles, but the important rule is that particles may not overlap. A good model looks like,

$$E_{ex} = \begin{cases} +\infty & r_{ij} \le r_1 \\ 0 & r_{ij} > r_1 \end{cases} \quad (1)$$

where r_{ij} is the distance between particles and r_1 reflects the size of the particles and E_{ex} is the energy for exclusion (overlap).

Graphically:

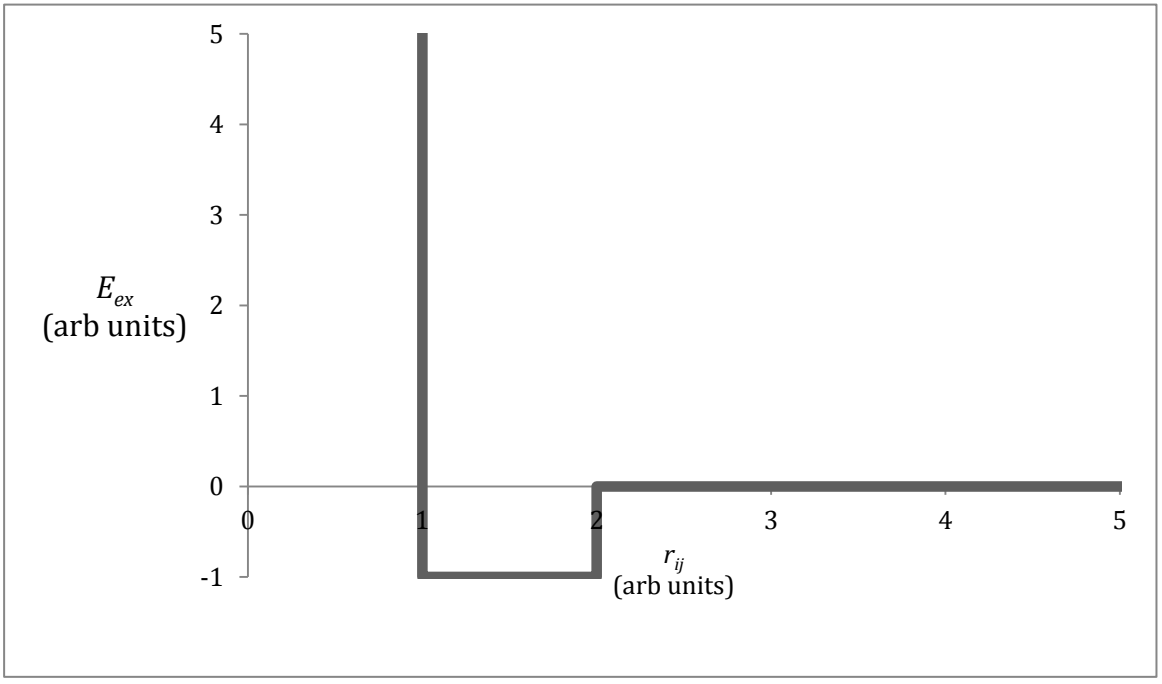


3.1.2 Lennard-Jones / Noble gas / excluded volume

Even in simple uncharged systems, there are weak attractive forces between the particles. These are weak and short-range:

$$E_{ex} = \begin{cases} +\infty & r_{ij} \le r_1 \\ -c & r_1 < r_{ij} < r_2 \\ 0 & r_{ij} \ge r_2 \end{cases} \qquad (2)$$

where r_{ij} is the distance between particles *i* and *j*. You could also say,



Both of these forms are discrete approximations to a continuous form $E_{ex} = 4\epsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{-6}\right)$. We use the discrete form because it is fast to calculate.

3.1.3 Particles with charges

Imagine we are simulating Na^+ and Cl^-. We have exclusion, as before, but now we add a coulombic term so,

$$E_{ex} = \begin{cases} +\infty & r_{ij} \le r_1 \\ -c_1 + \frac{q_i q_j}{c_2 r_{ij}} & r_1 < r_{ij} < r_2 \\ \frac{q_i q_j}{c_2 r_{ij}} & r_{ij} \ge r_2 \end{cases} \quad (3)$$

and you can draw a graph of this. c_2 is a constant which you could call the dielectric constant or $4\pi\epsilon_0$.

This model is similar to the real world. Particles with the same charge repel each other. Particles with opposite charges attract each other, but there is always a stronger repulsion at short distances, otherwise oppositely charged ions would collapse on to each other.

This interaction is continuous (not discrete) and will be slower to calculate.

3.2 Dynamics

There are different ways to model motions. Sometimes, particles have a memory. Their velocity at time $t + \delta t$ depends on their velocity at time t. That would be a good way to model planetary motion. Our particles have velocities, but no memory of them. This is the classic model for systems such as colloids where the displacements are determined more by collisions than by inertia.

We will make a slight simplification. We will move one particle at a time.

Systems in the world usually follow a Boltzmann distribution. The probability p_i of state *i* is $p_i = Z^{-1}\, e^{\frac{-E_i}{kT}}$ where E_i is the energy of state *i* and Z is the partition function. k is the Boltzmann constant and T is the temperature. We choose units of temperature so we can pretend $k = 1$ or simply say, $p_i = Z^{-1}\ e^{\frac{-E_i}{T}}$. Temperature will not be important in the first version, but it will be very important later.

In the monte carlo method, one makes random moves and accepts them with a probability from some distribution. In physics and here, this is usually the Boltzmann distribution. The partition function might be important, but you cannot calculate it, so you cannot calculate p_i. This is not a problem, if you have two states, you can calculate the relative probabilities, $\frac{p_i}{p_j}$. This means we have some starting coordinates with a probability p_{old} and some trial coordinates with a probability p_{trial}. The ratio of probabilities is

$$\frac{p_{trial}}{p_{old}} = \frac{e^{\frac{-E_{trial}}{kT}}}{e^{\frac{-E_{old}}{kT}}} = e^{\frac{E_{old}-E_{trial}}{kT}} \tag{4}$$

but we said we would use units so that k disappears and we define $\Delta E = E_{old} - E_{trial}$. Then we can write

$$\frac{p_{trial}}{p_{old}} = e^{\frac{\Delta E}{T}} \tag{5}$$

This leads to the Metropolis Monte Carlo Method.[1]

4 Methods

The most important method is metropolis monte carlo. In pseudo-code:

[1] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N. and Teller, A. H., Equation of State Calculations by Fast Computing Machines. J. Chem. Phys., 1953. 21, 1087-1092.

set *max_steps*

set *t* to temperature

store initial coordinates c_{old}

store total energy in E_{old}

while (*nstep* < *max_steps*) {

move_coordinates_with_random_step

calculate E_{trial} and ΔE

if $\Delta E > 0$

calculate $r = \frac{p_{trial}}{p_{old}}$ from eq. 5

else

$r = 1$

if ($r \geq$ rand (0..1)) {

save coordinates to c_{old}

$E_{old} = E_{trial}$

$n_{accepted}$++

}

nstep++

}

5 Implementation

5.1 Standard library calls and documentations

Use standard library calls. You can find the documentation for all standard functions by typing **`man drand48`** / **`man getopt`** / **`man whatever`**. Do not use linux specific functions. They are not necessary and only make your programs unportable. Functions from the standard library are available on machines ranging from linux to windows.

Try to write robust code. Check the return value from every call to functions like **`malloc()`** or **`fopen()`**. Check that your program is given the correct number of arguments (check the value of **`argc`**). If your program is unhappy, print out the reason and stop with **`exit (EXIT_FAILURE)`**.

5.2 Random numbers

You will need random numbers. When you want a number between 0 and 1, use **`drand48()`**. When you want an unsigned integer between 0 and *n*, use

`my_int = lrand48() % n`.

Use **`srand48()`** to seed the random number generator. Do not use the time of day or anything you think is random. In numerical work, one always uses a controlled series of random numbers so you can reproduce a calculation and debug.

6 Tasks

It is easiest to have three programs.

1. generate coordinates
2. run a simulation
3. analysis

We will consider step 3 later.

6.1 Generate coordinates

We need a program for generating initial coordinates. This will let one try out different ideas and starting points. For today, we will consider two variations, give below. The program must take the following arguments:

1. Number of particles (*n*)
2. *x* dimension (size in arbitrary units)
3. *y* dimension (size in arbitrary units)

Given the arguments, your program should place *n* particles within the box and write coordinates to a file.

6.1.1 Format

At the moment, all particles are the same. Let us start with a format like

```
nnnn
x1 y1
x2 y2
...
```

Where the first number, **nnnn**, is the number of particles in the file. Later, we can add more columns with properties of the particles, such as charge.

6.1.2 Random coordinates

Just place *n* particles in the box randomly.

6.1.3 Uneven distribution

Place *n* particles in the box, but place all of them in the first 10 % of the *x* coordinate.

6.1.4 Testing

We will not worry about collisions in the initial coordinates. You should check that your program works when you

- have 0, 1, 10^4 particles
- have $x_{max} = 10\ y_{max}$ and $x_{max} = \frac{y_{max}}{10}$

You should be happy with this part of the code before you go on. If you have problems at this stage, you will have worse problems in two or three weeks.

The format above lets you quickly check if the coordinates appear to be sensible. Since it is an *xy* file, you can make a scatter plot in gnuplot/xmgrace/your favourite plotting program. Obviously, there should be no points outside of the regions you picked.

6.2 Simulation

Let us hardwire the size of the particle (r_1) in eq. 1 and set it to 1.0 units. Although it would be more elegant to put in a command line option which lets you specify r_1.

You have to write a simulation program which implements the collision avoidance in eq. 1. As you are writing, remember that we will move to the other interaction forms (eq. 2 and eq. 3) over the next few weeks. The program arguments should be

- number of steps
- temperature
- input coordinate file
- how often to write out coordinates
- optionally – seed for the random numbers

The program should

- read the initial coordinates, simulate according to the pseudo code on page 5. Every *m* steps, it should write out coordinates.
- Include code to write out the energy every step

Problem for you to solve:
The energy form given by eq. 1 has an infinity when particles overlap. Once the system is at equilibrium, this is not a problem. Any move which leads to overlapping particles will be rejected. The starting coordinates are not at equilibrium and may have overlapping particles. This means, that at the start, you will move particles. The result you want is,

if the move introduces a new overlap
    the energy goes up by ∞ and is rejected
if the move decreases the number of overlapping particles
    the energy goes down and the move will be accepted
if the number of overlapping particles does not change
    the move will be accepted

The problem is, that the ∞ is very artificial. Ask yourself if you really need +∞, or if a large positive number is sufficient. What is the danger with replacing +∞ with a large positive number ?

You will be left alone with the programming, but before you start, we will discuss

- ways to calculate energy
- structuring the code so it can be extended
- the parameters you will need to move between different score functions