UMEÅ UNIVERSITY
Department of Physics
Modeling and Simulation, 7.5hp

September 30, 2010

Molecular Dynamics, Stochastic simulations, and Monte Carlo

Peter Olsson

Miscellanous comments

- Send the report as a mail to `Peter.Olsson@tp.umu.se` with subject "ModSim lab 5", no later than Monday October 18, 08:00. Corrections may, however, be handed in after that deadline.
- Provide a path to the relevant code on the computers, or make it available by some other means.
- Note that there are three voluntary excercises that may give bonus points.
- The program uses several functions from the C library (like `strcmp` and `strchr`). Information on them are best found in the `info` system.
- There are routines in ran.c for getting different kinds of random numbers.

1 Introduction

A gas of particles is a very important system in physics. If the density is low, one can neglect collisions between particles. When that is a good approximation we say that the gas behaves like an ideal gas and is then well described by the ideal gas law,[1]

$$pV = Nk_BT,$$

which is a relation between pressure p, volume V, number of particles N, temperature T, and Boltzmann's constant, $k_B \approx 1.38 \times 10^{-23}$ Joule/Kelvin.

1.1 Interaction

The fact that gases liquify at higher densities is due to interactions between the molecules. This interaction is repulsive at short distances and attractive at somewhat larger distances. This is commonly modelled with the Lennard-Jones interaction

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{1}$$

Here the length σ and the energy ϵ are material parameters. We will however use units such that they are both equal to unity. The force between the particles is given by $F = -dV/dr$; a positive sign means a *repulsive* force.

$$F(r) = \frac{\epsilon}{r} \left[48 \left(\frac{\sigma}{r}\right)^{12} - 24 \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{2}$$

The above expression gives the magnitude of the force; to capture the vector nature we write

$$\mathbf{F} = -\nabla V = -\frac{dV}{d\mathbf{r}} = -\frac{dr}{d\mathbf{r}}\frac{dV}{dr} = -\left(\frac{dr}{dx}, \frac{dr}{dy}, \frac{dr}{dz}\right)\frac{dV}{dr} = \left(\frac{x}{r}, \frac{y}{r}, \frac{z}{r}\right) F(r).$$

The last equality follows from differentiation of $r^2 = x^2 + y^2 + z^2$ which gives

$$r\ dr = x\ dx + y\ dy + z\ dz.$$

[1] The ideal gas law is also often written $pV = nRT$ with the number of moles n and the gas constant R.

1.2 Molecular dynamics

The dynamics of our system with N particles is governed by two coupled differential equations:

$$\begin{aligned} \dot{\mathbf{r}}_i &= \mathbf{v}_i, \\ \dot{\mathbf{v}}_i &= \mathbf{F}_i, \end{aligned}$$

where $\mathbf{F}_i$ is the sum of the forces acting on particle i due to the interaction with the other particles. In the simulations this should be integrated with a small but finite time step, Δt, and with the notation $\mathbf{v}_n = \mathbf{v}(n\Delta t)$ the leap-frog method becomes

$$\mathbf{v}_{n+1/2} = \mathbf{v}_{n-1/2} + \mathbf{F}_n \, \Delta t, \tag{3}$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_{n+1/2} \, \Delta t. \tag{4}$$

2 A program for Molecular dynamics and Langevin dynamics (mandatory part)

We are here going to simulate an interacting particles in two dimensions. An important reason for looking at this problem in two dimensions is that it then becomes much easier to visualize the configurations (which is one of the voluntary exercises).

Your task is now to complete an almost finished program.[2] The first step is to look through the program *carefully* to understand how it is meant to work. Check both the `Makefile` and all the source files. The program begins (as always) in the `main` function which is at the bottom of `lang.c`. This is also an opportunity to learn a few programming tricks.

2.0 Getting started

Quite a few files are needed to complete this lab. To get them all together with some new directories to your home directory, create a new directory for

[2]Some of you would prefer instead writing everything from scratch and would perhaps also do that successfully. This exercise could anyway be a good one since there is a good chance that you will be asked to modify existing programs in your coming professional activities.

this lab, and `cd` to that directory. Open a terminal[3] and execute[4]

```
tar xzf /home/olsson/md-mc.tgz
```

which is the command to e**x**tract the content from the g**z**ipped (compressed) tar **f**ile. The same files is also available at `http://www.tp.umu.se/ModSim/-files/md-mc.tgz`. Follow the steps below to convince yourself that your program is correct. In the source code the incomplete sections are marked with "`Fix this (1)`", where the numbers, 1–5, correspond to the numbers below. This part is only to become familiar with the workings of the program and should not be included in the report.

1. Complete the function `measure` (in `common.c`) which is necessary to perform the measurements in the program. Compile the program (type `make lang` in the terminal) and execute it in a shell by typing

   ```
   ./lang N=50 L=14 T=1.0 read=conf/start50
   ```

   the program should now give the energies (potential and kinetic) for the starting configuration: `Energy = 0.292288 (-0.70007 0.992358)`.

2. Next, remove the `exit(0)` in `lang.c` that killed the program. Study the function `step` and then complete the functions `one_force` and `vel_from_forces` in `common.c` to implement the leap-frog method. To do this you will have to look through the program (mainly common.c) carefully to understand how it is intended to work.

   Note that the default for the friction is $\alpha = 0$ which means that the program by default does molecular dynamics in spite of its name. With the same command line as before you should now get the following output for the positions and velocities for particle number 0:

[3] Use the left mouse button to select `Applications` on the very left of the top panel and then `Accessories` where `Terminal` is found at the bottom. It could also be convenient to make the terminal directly accessible from the panel. To do that press the *right* mouse button at the `Terminal` symbol and choose `Add this launcher to the panel`.

[4] In computing, tar (derived from tape archive) is both a file format (in the form of a type of archive bitstream) and the name of the program used to handle such files. ".tgz" is the same as ".tar.gz" – a gzipped tar file.

```
rx ry vx vy = 5.26728  1.64202  -0.536492  -1.01115
```

3. Remove the print statement for `rx ry...` and the `exit(0)` statement and make a loop to do `par->ntherm` calls for thermalization to the function `step` instead of the single call.

4. We want both averages and standard errors from the collected data and that is prepared for in the code. Note the meaning of `v0[]`, `v1[]`, and `v2[]` and complete the function `print_standard_error` (in `lang.c`) to print out the mean and the estimated standard error according to the following formulas:

$$X = \frac{1}{\text{nblock}} \sum_i x_i, \quad X_2 = \frac{1}{\text{nblock}} \sum_i x_i^2, \quad \text{stderr}(X) = \sqrt{\frac{X_2 - X^2}{\text{nblock} - 1}}.$$

5. The program should now perform Molecular dynamics. To change this to Langevin dynamics you need to complete `langevin_forces` (`common.c`) and calculate the magnitude of the noise (at the bottom of `lang.c`) correctly. See Sec. 2.2 for details. Note that $\langle \zeta^2 \rangle = 1/3$ if ζ is from a uniform distribution between -1 and 1. Adjust the magnitude of the noise and run the program with $\alpha = 0.1$ to check that the kinetic energy per particle is equal to T, which is $T/2$ per degree of freedom (= dimension).

The program should now be ready for the two following exercises.

2.1 Molecular dynamics and the time step

In molecular dynamics it is important to check that the time step is not too big. To that end one looks at the total energy which should be a constant. Do a number of runs, starting from `conf/start50`, with the time steps, $\Delta_t = 0.001$, 0.005, 0.01, and 0.02. Use at least 20 blocks (`nblock=20`) at the larger Δ_t. The energy data is stored in files in directory `efile`.

> **Exercise 2.1:** Show figures for total energy vs. time and comment on the results.

2.2 Langevin dynamics – the effect of a finite α

In Langevin dynamics the total energy fluctuates and it is interesting to examine how these fluctuations depend on α. In this dynamics both damping and noise give changes to the velocity:

$$m\mathbf{v}_i(t+\Delta_t) = m\mathbf{v}_i(t) + (\mathbf{F}_i - \alpha\mathbf{v}_i + \boldsymbol{\zeta}_i)\Delta_t. \tag{5}$$

Here $\boldsymbol{\zeta}_i$ is a vector noise where each component has the magnitude

$$\langle\zeta^2\rangle = \frac{2\alpha T}{\Delta_t}.$$

> **Exercise 2.2:** Do runs with $\alpha = 0.01$, 0.1, and 1.0. Use the default value of the time step, $\Delta_t = 0.01$ and plot energy vs. time. To interpret the results it might be good to recall that m/α has the dimension of time.
>
> Also plot the velocity correlation function $g_v(t)$ (from the files in directory `vcorr`) for $\alpha = 0.0$, 0.1, and 1.0. Comment on the results.

3 Brownian dynamics and Monte Carlo

Write a new main program e.g. `brown.c`, which implements Brownian dynamics. This may by done by starting from `lang.c`. The dynamics is then given by

$$\mathbf{r}_i(t+\Delta_t) = \mathbf{r}_i(t) + \mathbf{F}_i\frac{\Delta_t}{\alpha} + \boldsymbol{\eta}_i\Delta_t,$$

where each component of the noise is characterized by

$$\langle\eta^2\rangle = \frac{2T}{\alpha\Delta_t}.$$

Check that the program is correct by comparing the obtained potential energy when using $\alpha = 10.0$ and $\Delta_t = 0.001$ to the results from the Langevin program.

From this version it is then easy to make another program that does Monte Carlo. You can either make a new file `mc.c`, or use the same file for both programs and make use of preprocessor directives to select parts of the code:

```
#ifdef MC
  something for Monte Carlo
#else
  something else related to Brownian motion;
#endif
```

The advantage with a single file is that additions to one of the programs (e.g. measurements of the pair correlation function) immediately become available to both versions. On can then handle this by compiling with (or without) `-DMC` which corresponds to putting `#define MC` in the file. (In the Makefile this is best handled through `CPPFLAGS = -DMC` which sends flags to the preprocessor.[5])

The relevant changes to implement Monte Carlo will mainly be to write a function that performs a Monte Carlo sweep. Note that you also should keep track of the acceptance ratio and that you will need a parameter that specifies the maximum distance to move the particles.

One Monte Carlo sweep is done by looping over the particles i and for each i do the following operations:

1. Suggest a new position: $\mathbf{r}_i' = \mathbf{r}_i + b\boldsymbol{\delta}$.
2. Calculate the energy difference $\Delta E = E_i' - E_i$ and
$$\alpha_{i\to i'} = \min\left(1, e^{-\Delta E/T}\right).$$
3. Accept this new position with probability $\alpha_{i\to i'}$, i.e. generate a random number ξ and accept if $\xi < \alpha_{i\to i'}$.

3.1 Comparison

We will now demonstrate that Brownian motion gives the same results as Monte Carlo in the limit of small Δ_t/α:

> **Exercise 3.1:** Make a long run with the Monte Carlo program to get good precision for the potential energy with the same parameters as before, $N = 50$, $L = 14$, and $T = 1.0$. Run

[5]This naming is a bit confusing. CPP stands for C preprocessor; for the flags to the C++ compiler one uses CXXFLAGS, see the `make` documentation. However, in practice this seldom matters since the preprocessor is usually run immediately before compilation.

the Brownian dynamics program with the following parameters: $(\alpha, \Delta_t) = (10.0, 0.001)$, (10.0, 0.002), (10.0, 0.003), and (10.0, 0.004). Plot the potential energy *with error bars* vs. Δ_t/α; put the Monte Carlo data at $\Delta_t/\alpha = 0$. Be sure to run long enough to get reasonably precise data, which in this case means that the statistical errors ought to be < 0.005. (It should be possible to see a clear trend in the data.)

4 Voluntary exercises

4.1 Different phases

At high temperatures (here around or above $T = 1$) the system is a gas with a uniform density. When the temperature is lowered the attractive interaction starts to become more important and the particles prefer to be close to one another. This will then lead to a denser region (a liquid) coexisting with the region with lower density (a gas). The liquid is however about as disordered as the gas. If the temperature is lowered even further the system will try hard to get the lowest possible energy, which is obtained by forming an ordered crystal. The transition between such phases may be studied with advanced methods but we will here just look at the snapshots in the stored configuration files.

Exercise 4.1: Do Monte Carlo at temperatures $T = 1.0$, 0.9, 0.8,... 0.1 on a larger system with $N = 200$ and $L = 28$. Be sure to start the run at the lower temperature from the configuration of the next higher one. Show representative pictures of (1) gas, (2) gas-liquid coexistence, and (3) solid phase (well, gas-solid coexistence), and try to approximately locate the phase transition temperatures. (1p)

4.2 Correlation functions

It is interesting to see if/how the different phases and transitions are seen in the pair correlation function.

Create the files `pcorr.c` and `pcorr.h` to implement the functions needed for measuring the histogram of particle distances. Note that many parts may be taken over from `vcorr.c` though the present case is considerably simpler

since there is no need to store data from earlier times. Use the histogram data to calculate the pair correlation function.

> **Exercise 4.2:** Start from the stored configurations at the respective temperatures and run long enough to get nice curves for the pair correlation function. Is there a difference between gas and gas-liquid coexistence in the pair correlation function? In the solid there are some more peaks. Try to explain their origin. Again, show representative figures for the three regions. (2p)

4.3 Visualization

It is of course possible to make nice 3D pictures of some spherical particles which move around in the simulations. That may be good for a nice presentation, but we are here mostly interested in visualization as a means of seeing what is going on, e.g. for debugging the program. When that is the purpose one should go for some simple functions that are easy to put into the program, as the g2 library. (For a quick introduction, see the linux guide, `http://www.tp.umu.se/ModSim/linux-guide`.)

> **Exercise 4.3:** Return to the program with Langevin dynamics and implement graphics on the screen (i.e. X11) with the g2 library. Provide a path to an executable file in the computer system. (2p)