ASSIGNMENT 7: SOLUTION

E. ARNOLD, M-S. LIU, R. PRABHU & C.S. RICHARDS
THE UNIVERSITY OF CAMBRIDGE

Written in XeLaTeX

MOLECULAR DYNAMICS

INTRODUCTION

It is a common task to need to model the time evolution of a set of particles. In surface physics, the conceptual interpretation of this is simple; a set of molecules can be individually tracked as they interact with the surface, and with each other. Of course, with the involvement of so many entities, the reality is much less straightforward. Providing a computational model requires a great many ideas, and significant processing power: gradually learning this skill, the reader should exert significant caution - it is very difficult to write a fast script, especially without errors!

PREREQUISITES

From the *Theory Handbook,* the chapters on:

- The Fourier transform,
- Verlet integration,
- Langevin mechanics.

Ensuring that you understand the above before starting is vital. This document does not cover them in detail. We recommend that the reader understands the relation between the van Hove pair correlation function and the intermediate scattering function before starting this tutorial, allowing them to make links to direct measurements with spin echo techniques.

THEORY: LANGEVINS

There is a surprisingly simple equation that dictates the motion of an adsorbate on a substrate. Known as the *Langevin*, this equation has been extensively studied in the literature. The simplified form of the Langevin equation, applying under a particular set of assumptions, is:

$$m_i \ddot{\mathbf{r}}_i = -\eta m \dot{\mathbf{r}}_i + \xi(t),$$

where:

1. $\mathbf{r}_i$ is the position of the ith adsorbate,

2. m_i is the mass of the ith adsorbate ,
3. η is a friction coefficient,
4. ξ is a stochastic force, described by a vector quantity.

What does this equation represent? As you may reasonably expect, the motion of a particle on a surface is *not* uniform. The most simple interpretation of this is that the particle experienes a force: consequently causing it to accelerate, and change direction.

Assumptions

When can we use this form of the Langevin equation? Noting the fuller form of the Langevin given in the *Theory Handbook*, we notice that two terms have disappeared: the gradient of the potential, and the adsorbate-adsorbate interactions. We can easily justify removing these terms, in suitable contexts. For weakly corrugated surfaces, the potential will change very little around the surface (thus justifying the removal of the gradient). For sparsely populated adsorbates, they are simply too far away to strongly affect each other - and so feel a very small force.

The Stochastic Force

The stochastic force ξ is a more abstract term in the Langevin. For a detailed discussion of its origin, see the *Theory Handbook*. For the purposes of this document, we will repeat what has already been mentioned: the stochastic force is normally distributed in each direction, and independent in each direction. In a small time interval δt, the impulse it exerts (in each direction) has a standard deviation

$$\sigma = \sqrt{2k_B T_s m \eta \delta t}.$$

This is the impulse! Do not get it mixed up with the force.

There is one further comment to make. Why does the stochastic force take this value? This standard deviation is the precise value that allows the system to maintain thermal equilibrium. If it were any different, thermal equilibrium would not be maintained. This would not resemble a steady state for the system!

THEORY: THE EQUIPARTITION THEOREM

This section includes additional theory required to complete this task. For a system with n quadratic modes, all of which are degrees of freedom, the expectation value of the energy of the system is:

$$\langle E \rangle = \frac{n}{2} k_B T,$$

where k_B is the Boltzmann constant, and T is the temperature of the heat bath to which the system is connected.

What do "degrees of freedom" mean? Simply put, if a system has n quadratic degrees of freedom, then we can write the energy of the system in the form:

$$E = \sum_{i=1}^{n} \alpha_i x_i^2,$$

where each x_i is a distinct variable. Consider how many degrees there are, for a particle moving on a surface. Is it the same for both atoms and molecules (consider the form of the potential function of a covalent bond)?

OBJECTIVE OF THIS TUTORIAL

In this tutorial, we will attempt to build a primitive model of atoms moving on a surface. This surface will have very weak corrugation, allowing us to massively simplify the system. We will also neglect forces between molecules, thus simplifying the Langevin further. The result of these assumptions is the form of the Langevin given above. We will numerically solve the system using Verlet integration.

Doing Molecular dynamics is a little more involved than the other tasks set so far. Don't worry if it takes several attempts to get it working properly! This is entirely normal.

TASK

1. The main task set by this tutorial is straightforward to describe. Given a single particle was at rest at the origin at $t = 0$, solve the Langevin equation, using Verlet integration. Plot the two dimensional time evolution of the position of the particle on the surface.
2. Repeat the simulation above, except for an ensemble of 50 particles. Plot each particle on the same graph, in a different colour.
3. State how many quadratic degrees of freedom there are for a diffusing particle on this surface. Using the equipartition theorem, state what the equation for the total energy of the adsorbate is, as a function of temperature.
4. Find the root mean square kinetic energy of the particles in this ensemble, as a function of time. Plot this as a graph. Confirm that this corresponds to the temperature of the substrate surface. Provide a physical interpretation of this correspondence.

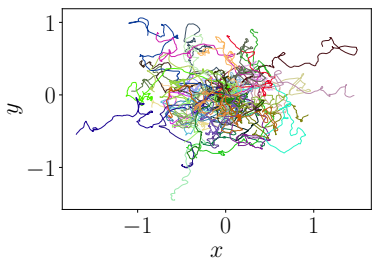

Figure 1 | An ensemble of 50 particles moving on a surface, using the model in this document. The units on the axis are arbitrary

Parameters

Throughout this tutorial, we advise that you set the parameters as below.

1. The friction coefficient η as 50ps^{-1},
2. The mass of each particle m as 1amu,
3. The temperature of the substrate T_s as 100K,
4. The Boltzmann constant k_B as $0.8314\text{Å}^2\text{amu} \cdot \text{ps}^{-2}\text{K}^{-1}$,
5. The timestep δt from the Verlet algorithm as 0.001ps,
6. 100 000 steps in the Verlet algorithm.

SUMMARY

This tutorial introduces the skill of how to model the diffusion of particles on a flat surface, using the Langevin equation of motion. This naturally leads on to more advanced simulations.

SOLUTIONS

MOLECULAR DYNAMICS: MAIN TASKS

Task 1: Question

The main task set by this tutorial is straightforward to describe. Given a single particle was at the origin at $t = 0$, solve the Langevin equation, using Verlet integration. Plot the two dimensional time evolution of the position of the particle on the surface.

Task 1: Solution

The aim of this task is to solve the Langevin equation of motion. For a weakly corrugated surface, with few adsorbates, the Langevin can be written as

$$m_i \ddot{\mathbf{r}}_i = -\eta m \dot{\mathbf{r}}_i + \xi(t).$$

If you are not familiar with the symbols used in this equation, see the description in the body of the text. Our first objective must be to initialise all of the variables relevant to the molecular dynamics simulation. This is fortunately very easy! The listing for this is given below.

```
%-------------------------------------------------------------------
% initialise variables
%-------------------------------------------------------------------

% friction coefficient, units of 1/picoseconds
eta=50;

% mass of the diffusing particle, units of amu
mass=1;

% temperature of the substrate surface, units of Kelvin
Ts = 100;

% Boltzmann constant, units of A^2 amu ps^-2 K^-1
Kb = 0.8314;

% timestep used in the simulation (and integration), units of
    picoseconds
dt = 0.001;
%-------------------------------------------------------------------
%-------------------------------------------------------------------
```

Next, we note that computing the square of the timestep beforehand, as we will require for Verlet integration, will negate the need to calculate it again at each step. This is:

```
% Calculating the square of the timestep advance will slightly accelerate the program
dt2 = dt^2;
```

Recall that the standard deviation of the impulse per unit time due to the stochastic force is

$$\sigma = \sqrt{2k_B T_s m\eta \frac{1}{\delta t}}.$$

Then it is clear that this standard deviation can be expressed as:

```
%-----------------------------------------------------------------
% stochastic force
%-----------------------------------------------------------------

%  the standard deviation in the impulse per unit time from the stochastic force (i.e. the "force" it exerts)
standarddev = sqrt(2/dt * eta * Kb * Ts * mass);
```

If you have worked through the section on verlet integration in the *Theory Handbook*, then the remainder of task 1 is trivial. In anticipation of needing to have to use the stochastic force in the simulation, we know that we wil need to know the force at any given time; it is more effective to generate an array of values for the stochastic force developed during each timestep in one go, rather than "on the fly". First, we must specify the size of the array with:

```
% number of dimensions
ndim=2;


% number of timesteps in the simulation
nstep=1000000;
```

Then the array for the stochastic force can be generated trivially with the code:

```
% The impulse is the product of the standard deviation, and a normally distributed sample.
% This sample takes the form of a two dimensional array of values
% all of which are normally distributed
% The array is nstep x ndim in size
% ndim is the number of dimensions (two: x and y), nstep is the number of steps in the simulation
force = standarddev * randn(ndim , nstep);

%-----------------------------------------------------------------
%-----------------------------------------------------------------
```

What do we do now? We have all the variables we need to perform Verlet integration. To proceed, we require an arrary of positions and velocities to fill. These can be initialised with:

```
44 %-------------------------------------------------------------------
45 % initialise position and velocity
46 %-------------------------------------------------------------------
47
48 % create two vectors of nan ("not a number"), which are the same
       size as the stochastic force.
49 % these contain both the x and y directions!
50
51 % the velocity as a function of number of steps
52 v = ones(ndim ,  nstep).*nan;
53
54 % position as a function of number of steps
55 r = ones(ndim ,  nstep).*nan;
56
57 %-------------------------------------------------------------------
58 %-------------------------------------------------------------------
```

This is not quite everything we need to perform the integration. We also require the boundary conditions! These are:

```
59 %-------------------------------------------------------------------
60 % boundary conditions
61 %-------------------------------------------------------------------
62
63 % particle starts at the origin
64 r(:,1)=zeros;
65
66 % particle starts at rest
67 v(:,1)=zeros;
68
69 %-------------------------------------------------------------------
70 %-------------------------------------------------------------------
```

Integrating the equation using Verlet integration yields the trajectory of the particle. The numerical integration is trivial. The listing for this is:

```
71 %-------------------------------------------------------------------
72 % do the verlet integration
73 %-------------------------------------------------------------------
74 % iterate through all of the timesteps. Start with the second,
       as the first timestep is the boundary condition.
75 % this is not 0 indexed!
76
77
78 % i is an arbitrary iterator
79 for i = 2 : nstep
80
81         % Verlet integration: first calculate the next position,
               assigned the variable r
82         % r_1 = r_0 + v_0 dt + 0.5 a dt^2
83         % acceleration a is force F divided by mass m (i.e. -eta
               v_0 + stochastic force/m)
84
85         % new r is
86         r(:, i) =  r(:, i-1) + v(: , i - 1) * dt + 0.5 * dt2
               .*(-1 * eta.* v(:, i - 1) + force(:, i - 1)/mass );
87
88         % then calculate the next velocity, assigned the
               variable v.
89         v( : , i ) = ( r( : , i ) - r( : , i - 1 ) ) / dt;
```

```
90      end
91
92  %-------------------------------------------------------------------------%
93  %-------------------------------------------------------------------------%
```

All that remains is to plot the figure. This is easily achieved; we anticipate a Brownian-like plot of the trajectory of one particle. An example of this plot is given in the margin.

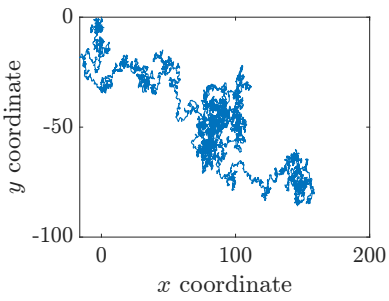

Figure 2 | A typical trajectory of a particle undegoing this diffusive regime.

Task 2: Question

Repeat the simulation above, except for an ensemble of 50 particles. Plot each particle on the same graph, in a different colour.

Task 2: Solution

Wrapping the code (from where the stochastic array is calculated to the end) in a for loop leads to the trajectory being calculated for a number of particles. We will not explore this here - the change is trivial.

Task 3: Question

State how many quadratic degrees of freedom there are for a diffusing particle on this surface. Using the equipartition theorem, state what the equation for the total energy of the adsorbate is, as a function of temperature.

Task 3: Solution

For a particle diffusing on a flat surface, there are two quadratic degrees of freedom. These are the kinetic energy along the x direction, and the kinetic energy along the y direction. Thus the energy is

$$\langle E \rangle = k_B T.$$

Task 4: Question

Find the root mean square kinetic energy of the particles in this ensemble, as a function of time. Plot this as a graph. Confirm that this corresponds to the temperature of the substrate surface. Provide a physical interpretation of this correspondence.

Task 4: Solution

This task is more complicated. We require generating the root mean square velocity for a collection of particles. Why do we do this? As the process is inherently random, the effect of noise is significant. Thus by

combining the effects of many different particles, the root mean square becomes closer and closer to the bulk mean. Thus as before, we start off with defining our variables

```
%-------------------------------------------------------------------
% initialise variables
%-------------------------------------------------------------------
% friction coefficient, units of 1/picoseconds
eta=50;

% mass of the diffusing particle, units of amu
mass=1;

% temperature of the substrate surface, units of Kelvin
Ts = 100;

% Boltzmann constant, units of A^2 amu ps^-2 K^-1
Kb = 0.8314;

% timestep used in the simulation (and integration), units of
    picoseconds

% Calculating the square of the timestep advance will slightly
    accelerate the program
dt = 0.001;

%-------------------------------------------------------------------
%-------------------------------------------------------------------


%-------------------------------------------------------------------
% stochastic force
%-------------------------------------------------------------------

%  the standard deviation in the impulse per unit time from the
    stochastic force (i.e. the "force" it exerts)
standarddev = sqrt(2/dt * eta * Kb * Ts * mass);

%-------------------------------------------------------------------
%-------------------------------------------------------------------
```

We will again require to generate the stochastic force. We note that the paramaters for the shape of the normally distributed sample are

```
%-------------------------------------------------------------------
% initialise a few extra variables
%-------------------------------------------------------------------

% number of dimensions
ndim=2;


% number of timesteps in the simulation
nstep=1000000;

%-------------------------------------------------------------------
%-------------------------------------------------------------------
```

To iterate over 50 particles, we need to perform the problem from task 1 50 times. This can be easily achieved with a `for` loop. However, we need to initialise something first! Our approach is to take a rolling

sum of the square value of the velocities. This effectively accumulates the total kinetic energy of the particles as the program works through the ensemble. Thus

```
47 %-------------------------------------------------------------------
48 % setup array to infer rms from
49 %-------------------------------------------------------------------
50
51 % Define a vector to accumulate the sum of the squares of the
       velocity of each particle, as a function of time
52 % This is a 0 vector until it is filled
53 % Each entry is the sum of the square velocity of all particles
       at each timestamp
54 sum_square_velocity_values = zeros(1, nstep);
55
56 %-------------------------------------------------------------------
57 %-------------------------------------------------------------------
58
59
60
61 %-------------------------------------------------------------------
62 % number of iterations
63 %-------------------------------------------------------------------
64
65 % Initialise number of particles, also the number of iterations
66 loops = 50;
67
68 %-------------------------------------------------------------------
69 %-------------------------------------------------------------------
```

We now have the most difficult section of the task. This is to produce the `for` loop relevant to the problem. Our approach is to use Langevin integration on the particles, one by one, and accumulating the square velocities as we go.

```
70 %-------------------------------------------------------------------
71 % do the verlet integration, and prepare array to find rms from
72 %-------------------------------------------------------------------
73
74 % iter is an arbitrary iterator - it is not used
75 for iter = 1:loops;
76     % calculate an array of values of the stochastic force for
           this iteration
77     force = standarddev*randn(ndim , nstep);
78
79     % initialise the velocity vector for this iteration
80     v = ones(ndim , nstep).*nan;
81
82     % initialise the position vector for this iteration
83     r = ones(ndim , nstep).* nan;
84
85     % boundary conditions: starts at rest at the origin
86     r(:, 1) = zeros;
87     v(:, 1) = zeros;
88
89     % This for loop performs the Verlet integration for each
           particle: returns the filled velocity and position
           vectors
90     for i = 2 : nstep;
91
```

```
92          % Verlet integration: first calculate the next position,
                assigned the variable r
93          % r_1 = r_0 + v_0 dt + 0.5 a dt^2
94          % acceleration a is force F divided by mass m (i.e. -eta
                v_0 + stochastic force/m)
95
96          % new r is
97          r(:, i) =  r(:, i-1) + v(: , i - 1) * dt + 0.5 * dt2
                .*(-1 * eta.* v(:, i - 1) + force(:, i - 1)/mass );
98
99          % then calculate the next velocity, assigned the
                variable v.
100         v( :, i ) = v(:, i-1) + (-1 * eta.* v(:, i - 1) + force
                (:, i - 1)/mass ) * dt
101     end
102
103    % Now that the vectors have been filled, add the squares of
           the velocities to the sum
104     sum_square_velocity_values = sum_square_velocity_values + v
           (1,:).*v(1, :) + v(2,:).*v(2,:);
105 end
106
107 %-----------------------------------------------------------------
108 %-----------------------------------------------------------------
```

All that remains is to calculate the root mean square velocity. This is easily achieved with the code:

```
109 %-----------------------------------------------------------------
110 % calculate rms at each timestamp, also calculate timestamp
111 %-----------------------------------------------------------------
112
113 % calculate root mean square velocity: divide by number of
        particles, then square root
114 rms_velocity = (sum_square_velocity_values/loops).^(0.5);
115
116 % Calculate the time stamps
117 time = linspace(0, nstep*dt, nstep)';
118
119 %-----------------------------------------------------------------
120 %-----------------------------------------------------------------
```

This data can be plotted, giving the figure in the margin. The temperature asympotically equilibriates to exactly half of the substrate surface temperature.

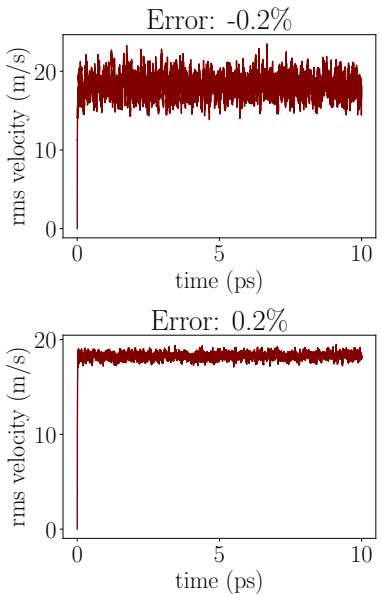

Figure 3 | The root mean square veocity of the ensemble, as a function of time. It quickly stabilises to a steady value. Top: 50 particles; bottom: 750 particles.