AtomicScaleSimulations_HW5

Questions, clarifications, comments, etc. about this homework can be posted on the course forum on Piazza.

Writing a Monte Carlo code

In this homework assignment, you will write a Monte Carlo simulation to calculate the transition temperature of Argon (as we did in homework 3 using molecular dynamics). You will be able to re-use many pieces of your code from HW 2 and 3 to accomplish this. An example code is also provided, although **you do NOT have to use it**. It is adapted from HW3 solution by removing all parts pertaining to velocity and acceleration - they are not needed by Monte Carlo. **Only mc_single_ptcl.py needs editing** to get a basic MC working. More coding is needed for force-biased moves.

> ***When reusing the old MD code, set box length to L=4.0.***

Potential Energy

We will start by writing a Metropolis Monte Carlo code. Only two central ingredients are needed for a correct Monte Carlo code: 1. calculate potential energy correctly. 2. make and accept moves in a way that satisfies detailed balance.

Remember, in classical Metropolis Monte Carlo you will be sampling configurations with probability
$\exp(-\beta V)$
, where V is the potential energy of the configuration and = 1/kT is the inverse temperature. This should be familiar to you as the Boltzmann distribution for a canonical ensemble at thermodynamic equilibrium.

To begin with, we should verify that we can evaluate the potential energy of the system correctly. You should already have code that does this from your molecular dynamics simulation.

To verify that your function works, calculate the potential energy of the initial configuration and verify that you get -231.570016481.

Numerical differences beyond the 3rd decimal place are acceptable. If you are using the example code, run the following unit test:

```
# setup system
pset = particleset.ParticleSet(64,48.0)
sc   = simulationcube.SimulationCube(4.0,pset)
pp   = pairpotentiallj.PairPotentialLJ(sc)

# initialize
pset.init_pos_cubic(sc.cube_length())
sc.update_displacement_table()

# compute initial energies
potential = compute_energy(sc,pp)

# unit test
assert np.allclose(potential,-231.570016481)
```

If your answer does not agree, make sure first that you are using the official initialization functions from HW2 and HW3.

Moves

Metropolis Monte Carlo works by proposing particle moves at random, and probabilistically accepting or rejecting these moves according to the Metropolis criterion (see below). We need to implement a function that updates the position of a single particle. You will update each particle position x by choosing the new position x'=x+ where is chosen from a Gaussian distribution of mean 0 and variance ^2, i.e. the probability of proposing a move to position x' such that = |x' - x| is

write a function to update the coordinates of a single particle

```
def update_position(pset,iat,sigma):
 #should return new position particle iat in pset
 # using the Gaussian distribution described above

    old_ptcl_pos = pset.pos(iat)

    # this is just a filler line, you have to implement the random move correctly
    new_ptcl_pos = old_ptcl_pos + np.zeros(*old_ptcl_pos.shape)

    return new_ptcl_pos
# end def update_position
```

The intended usage is

```
old_pos = pset.pos(iat) # save old position in case of rejection
new_pos = update_position(pset,iat,sigma) # propose
pset.change_pos(iat,new_pos) # move particle
sc.update_for_particle(iat)  # update distances
```

This implements the particle-by-particle move, where you move one particle at a time and accept or reject for every single particle move. It is also possible to make an all-particle move where every particle is moved at the same time and the entire move is accepted or rejected as a whole. All-particle move can take advantage of numpy vectorization to make the code much faster. However, without the help of forces, all-particle moves will almost always be rejected unless the step size (sigma) is tiny. If sigma is tiny, then random samples become correlated, reducing the Monte Carlo efficiency. Feel free to play around with all-particle moves, but I do not recommend it until forces are implemented.

Metropolis Acceptance Criterion

Now, let's calculate the acceptance probability. Remember, in Metropolis Monte Carlo, the probability of accepting a move from position x to x' is

where V(x) is the potential energy of the system in configuration x, and T(xx') is the probability of proposing a move from x to x'.

For the following 3 questions, you just need to answer them on paper.

1. For the Gaussian displacement move described above, write down expressions for T(x'x) and T(xx').
2. Show that, for this move, T(xx')/T(x'x) = 1.
3. Put these results together and write down an explicit expression for A(xx').

Metropolis Monte Carlo Algorithm

Now that we have the pieces, let's write the Monte Carlo loop. Remember the steps of doing a basic Metropolis Monte Carlo:

1. evaluate the potential energy of old configuration x
2. propose a move
3. evaluate the potential energy of new configuration x'
4. with probability A(xx')
   a. accept the move
   b. otherwise: reject (remember to return the particles to their old positions)

Monitoring Acceptance and Setting up Runs

When reusing the old MD code, set box length to L=4.0.

For all runs here, unless explicitly noted otherwise, we will use the following simulation parameters:

- Density = 1.0 (box length = 4.0)
- particle number = 64
- mass is 48.0
- Simulate the canonical ensemble at temperature T=2.0, or =1/T=0.5.

Before we set up some runs to generate data, we want to add functionality to our code that calculates acceptance ratios (i.e. measure the total number of accepted moves divided by the total number of attempted moves). Knowing the acceptance ratio of your MC simulation is particularly important to judge whether you are sampling configuration space efficiently. You can tune your simulation by adjusting value of sigma(), which sets the length scale for your attempted moves and thus determines the acceptance ratio of these moves. For the various values of step size (sigma), make 100 MC sweeps and report the potential energy trace vs. sweep and pair correlation function. A "sweep" means one renewal of all particle positions, so 100 sweeps is equivalent to 6400 single-particle moves for a system of 64 particles.

Note: In principle, statistics should be collected every acceptance/rejection. We throw out data by not outputting theses statistics. Output every sweep is OK for this problem because the correlation length is longer than 64 single-particle moves, thus we won't actually lose any statistics. You should check this for other Monte Carlo problems you may encounter.

1. Find a value of that gives you an acceptance ratio between 0.3 and 0.7. An acceptance ratio in this broad range is likely to give you decent sampling efficiency. Quote your acceptance ratio and your choice of . Using this value, perform a run of 100 sweeps. Produce a plot of the potential energy as a function of MC sweeps, and a plot of the pair correlation function for this run.

2. Find a value of that gives an acceptance ratio below 0.01; state the value of you are using. Run your simulation again for 100 sweeps and produce plots of the energy and pair correlation function.

3. Find a value of that gives an acceptance ratio greater than 0.95; state the value of you are using. Run your simulation again for 100 sweeps a nd produce plots of the energy and pair correlation function.

4. Comparing data from your three runs, discuss what happens to your simulation if your acceptance ratio is either effectively 0 or effectively 1. State specifically how this leads to an inefficient simulation in each case.

Now, using your choice of in the optimal range acceptance, perform some longer runs. Remember to **change your random seed** for each run.
5. Run the above simulation conditions for a minimum of 2000 sweeps.

a. Submit a trace plot of the energy as a function of Monte Carlo "time", i.e. the number of MC sweeps made.

b. State the total number of Monte Carlo steps, the average energy, and the standard error for your simulation.

c. Submit plots of the pair correlation function and structure factor.

6. Now do two simulations, one at temperature T=1.0 and one at a temperature below T=0.5. Produce plots of the energy, structure factor, and pair correlation function at each temperature.
7. Briefly compare these results with your data from the MD simulations in HW 3.

Detailed balance and smarter moves

This problem requires only pen and paper work. There is no programming required here. However, you should feel free to try implementing these more sophisticated moves, and you may wish to consider a semester project on this topic!

Standard Metropolis Monte Carlo Acceptance Criterion

Take the statement of detailed balance,

which says that the flux of particles from point x to x' is equal to the flux in the reverse direction, i.e. the net flux is zero. The following terms appear:

- (x) is the probability of the system being in configuration x
- T(xx') is the probability of proposing a move from x to x'.
- A(xx') is the probability of accepting the proposed move.

Show the standard Metropolis Monte Carlo acceptance criterion below satisfies the detailed balance equation.

Note this should only take a couple lines of algebra, but it is very important to understand where this equation comes from. Note also that this expression for A(x x') is not unique, but it is commonly used because of the symmetrical form between A(xx') and A(x'x).

Force-Bias Type Move

Above, we implemented a Gaussian displacement move that proposed a displacement drawn from a Gaussian distribution with mean 0 and specified variance ^2. In this case the forward and reverse sampling probabilities canceled out, i.e.

which you derived.

To see this, first write T(xx') as the probability of drawing the displacement (x' - x) from a Gaussian distribution centered on x with variance ^2:

$$T(x \to x') = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-(x' - x)^2/2\sigma^2\right\}$$

Because (x'-x)^2 = (x-x')^2, T(xx') = T(x'x).

For many more sophisticated moves, this is not the case. In fact, it is often non-trivial to find the reverse sampling probability, T(x' x), given a sampling probability T(xx'). We will study one case here.

The motivation of this move is to propose a displacement that is not random, but rather is biased in the direction of the force acting on the particle to be moved.
To implement this idea, we will make the same Gaussian displacement described above, but we will shift the center of our Gaussian in the direction of the force.
The center of the Gaussian will be shifted by

Thus the new move will choose a new position

$$x' = x + \eta + x_{\text{adjust center}}$$

where is chosen from a Gaussian distribution with probability

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\{-\eta^2/(2\sigma^2)\}$$

.

Again write a function to update particle positions, but this time use forces to guide your random walk

```
def update_position_force(pset,iat,sigma,force_on_iat):
 #return the new position particle iat in pset
 # as per described in the force-biased method
 return new_pos
# end def update_position_force
```

1. For this move,

   a. write an expression for T(xx').
   b. write an expression for the reverse probability T(x'x).
   c. combine these terms to write the acceptance criterion A(xx').

Finally, modify (but save your old code) your code to switch over to using this move.

2. Again, measure the energy of your system using the same number of sweeps as you did before.

   a. submit a trace of your potential energy vs. number of sweeps

   b. report standard error of your simulation

   c. compare the standard error you get using this move with the standard error you get using the more naive move.

   d. compare the total number of minutes it took to run.

3. Which move is better? Why?

Finally, let's look at a situation where such a move should be particularly effective, looking at a vacancy. To do this, you should take the following steps

- Figure out how to initialize your system with a vacancy in it (i.e. create a new position array that has one less particle and copy all but one of the particles in it).
- Add an observable that measures where your vacancy is. One effective method to do this is to define a number of lattice sites and say your vacancy is the lattice site further from any of the particles.
- Alternately using both moves, measure the approximate number of steps it takes for the vacancy to move from one site to another.

Your Own Move

Now we would like you to design your own move that you think would be more efficient than the conventional Gaussian displacement move we used above. The only requirement is that your move obey detailed balance, which you will demonstrate.

Describe your move in detail and explain why you think it will be more efficient.
For this new move, write an expression for the sampling probability T(rr').
Write the reverse probability T(r'r).
Write an expression for the acceptance probability A(rr').

Visualization (optional)

In this problem, we will visualize the system we are simulating with Monte Carlo.

> If you are trying this for the first time, please review the instructions here

Below, we have detailed information about writing the configurations from python using the correct syntax.

The particle set class has a method (append_frame) to dump configuration to file, a direct call will dump system configuration to "traj.xyz". You may specify a different configuration file by giving it a filename

```
pset.append_frame('mytraj.xyz')
```

The format for the output file must have the following properties:

- must end in ".xyz"
- the first line of each frame is an integer specifying how many particles there are: (i.e. 64)
- all other lines must be of the form: "i x y z", where "i" is a particle index or label. The first 64 lines are the coordinates for time step 1, the next 64 are for time step 2, etc.

For example, a file "traj.xyz" with 3 particles and 2 time steps would look like:

```
3
Ar 0.0 1.2 0.3
Ar 1.2 -0.2 3.3
Ar 3.5 0.2 1.1
3
Ar 2.2 2.4 0.5
Ar 2.2 -0.6 3.5
Ar 3.6 0.2 1.5
```

Once you have this file, you should be able to open it up in VMD by sourcing the Visualize.tcl file included in the example code. The default is to dump configurations every ndump=10 MC sweeps. You may change this in main.py.
You may also have to change:

- graphics representations drawing method to VDW
- sphere scale to approximately 0.1

You should then be able to click on the play button under VMD Main to see the movie.

Show a snapshot of a simulation for the liquid of the first frame of your simulation (which should look like a solid) and the last frame of your simulation (which should look like a liquid).