Physics 114 Statistical Mechanics Spring 2018

Presentation: Metropolis Monte Carlo

Intro:

We want to simulate a system which visits microstates that match up with a desired equilibrium macrostate. As the simulation proceeds, it should visit states $1, 2, 3..., i, ...$ such that the frequency of seeing microstate i is the same as the probability $P(i)$ drawn from the equilibrium probability distribution function (PDF). As we'll mention below, we might have to wait until some transient time has passed, so that *eventually* the system settles into visiting states with a frequency which reflects the PDF in a desired statistical ensemble. Here, we will talk about about the *Boltzmann PDF* :

$$P(i) \propto e^{-\beta E_i}$$

The set of states that are visited would thus be members of the *Canonical ensemble*. (We could alternatively do microcanonical, grand canonical, other ... ensembles. Monte Carlo is very flexible :-)

In the technique we describe below, we will generate a series of states, known as a *Markov Chain*. A necessary condition for convergence to the desired PDF is that when the system is in particular state, call it j, the next state k will be chosen so that

$$\frac{Probability\ j \to k}{Probability\ k \to j} = \frac{p_k}{p_j} = e^{-\beta(E_k - E_j)} \tag{1}$$

This is called the principle of *detailed balance*. It has been shown (Kalos and Whitlock, 1986) that the method developed by Arianna W. Rosenbluth, Marshall Rosenbluth, Augusta H. Teller, and Edward Teller (Metropolis et al, 1953) enforces both detailed balance + ergodicity (being able to visit every possible microstate, given enough computational time) . This will allow a system to eventually visit states with the desired, Boltzmann PDF.

Not to get off track, but generating states drawn from the correct PDF related to the idea of *Importance Sampling* . Suppose that we want the expectation value of an observable $< A >$, like energy $< E >$, or the fluctuations in energy $(E - < E >)^2 \equiv \Delta E^2$. (The latter will tell us the specific heat, since: $\Delta E^2 = kT^2 C_v$.) We could do

$$< A >= \frac{1}{N} \sum_{i=1}^{N} A_i P(i) \tag{2}$$

where in Eq. (2) states i are chosen completely at random. On the other hand, we could do

$$< A >= \frac{1}{N} \sum_{i=1}^{N}{}' A_i \tag{3}$$

where the restriction on the sum (the symbol $'$) in Eq. (3) means that the states i are chosen from the distribution $P(i)$. Using importance sampling via Eq. (3) to find averages is a little bit easier from a math point of view, and much more efficient than uniform sampling as in Eq. (2).

Method:

A Metropolis Monte Carlo (MMC) algorithm, as listed on G&T p. 227 accomplishes this. It is an accept/reject algorithm. The choice to go from $j \to k$ is accomplished in two stages.

- Trial step: Starting in j, state k is chosen at random.
- Accept/reject step: Decide if state k is accepted (k becomes the new state) or not (the old state j is the new state)

It is not necessary that in the trial step that the new state is completely randomly chosen. It can be for example within some "neighborhood" of the old state, and as long as all states are ultimately accessible, ergodicity will be satisfied. It is also not necessary that we use the specific accept/reject criterion mentioned below ... we'll generalize later. For now, here is the traditional MMC method:

1. Decide on a temperature. That is, let $1/kT = \beta$ and pick a value of β.
2. Initialize: Create an array of N numbers representing the energies of a set of N particles. Let them each start with the same energy, a random value for energy, ... this will affect how quickly the system converges to the equilibrium PDF, but otherwise, is unimportant.
3. Pick a random particle (That is, choose a random integer between 1 and N to determine your particle of interest.)
4. Trial move: Toss a "coin" (That is, choose another random integer, now either 0 or 1.) If heads, consider adding ΔE to the energy of the particle. If you are simulating an Einstein solid, you might always chose $\Delta E = 1$, but you can also choose a real number within some range. If tails, think about adding $-\Delta E$ to its energy (that is, subtracting energy).
5. Accept or reject the move:
   (a) Definitely accept the move if it would lower the particle's energy (but not below zero if the physical system, like an Einstein solid, cannot have negative energies.)
   (b) Accept the move with probability $p = e^{-\beta \Delta E}$ if it would raise the energy. To do this, you need to generate a real random number between 0 and 1, which G&T call r. If $r < p$, the move is accepted. Otherwise, it is rejected.

6. Go back to step 3 and iterate a lot of times ... you want each of the N particles to experience many moves.

Exercise: Show that this algorithm gives us detailed balance, Eq. (1).

Implementation:

Here is a snippet of Matlab code in the spirit of the Method above.

```
% Probability Distribution information
beta = 2.0;      % A target PDF parameter
pBoltz =  @(t) exp(-beta .* t);  % A  target PDF ... Boltzmann distribution
theta(1) = 0.3;  %initial value for the state of the particle

for i = 1:N
   theta_ast = proposal_PDF([]);       % Sampling from the proposal PDF
   alpha  = p(theta_ast)/p(theta(i)); % Ratio of probs at trial & orig. point
   if rand <= min(alpha,1)
      % Accept the sample with prob = min(alpha,1)
      theta(i+1) = theta_ast;
   else
      % Reject the sample with prob = 1 - min(alpha,1)
      theta(i+1) = theta(i);
   end
end
```

This implementation (which I hacked from the Web ... credit to Felipe Uribe-Castillo, 2011) is a little funky. We don't choose a random particle per se, but rather do a certain operation N times, treating each result as the new state of a random particle in the system. Those states migrate towards the desired Boltzmann distribution. Here, the state of the particle i, `theta(i)`, is considered to be its energy. The variable `theta_ast` is drawn from a "proposal PDF" ... for me, just a random number in the range $0, 10$. So this variable is the proposed new energy of the particle as in Step 4. The variable `alpha` gives the ratio of probabilities that we subject to acceptance test as in Step 5. If the move is accepted, the $i + 1^{st}$ particle takes on the new energy. If the move is rejected, the $i + 1^{st}$ particle takes on the energy of the i^{th} particle. Here are some figures from runs where $N = 5000$ and $N = 50000$.

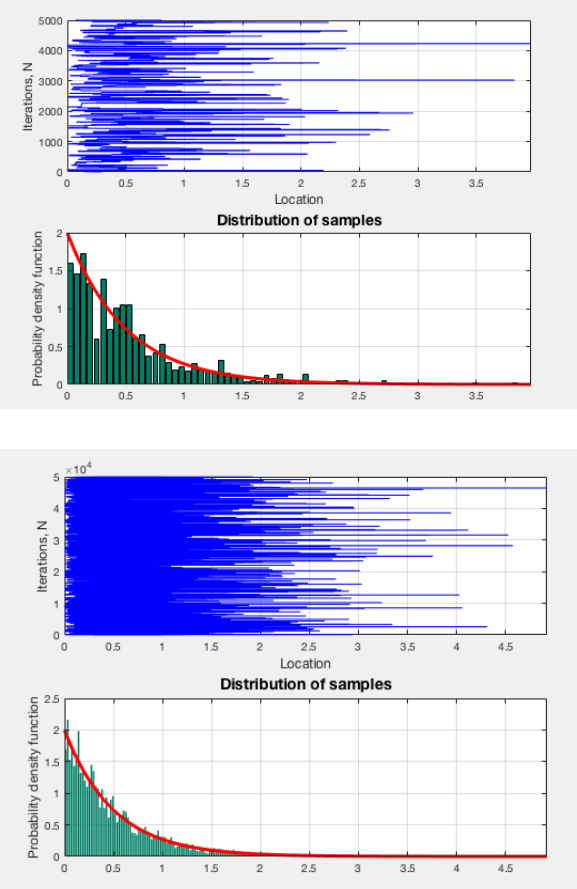

Figure 1: $N = 5000$ and $50,000$ steps of the MMC algorithm. We can see both the values of the energies (top part of each figure, in blue) and a histogram(bottom part of figure, green boxes) with the ideal probability distribution shown as a red curve.

Extensions:

We don't have to do the Boltzmann probability distribution. For example, my code does a uniform distribution (kind of silly, since we are using uniform random numbers to begin with) or a Gaussian:

```
pUniform = @(t) t.^0 ;   % A  target "PDF" ... uniform
nu = 5;   % A target PDF parameter
pGauss  = @(t) exp(- .5 * (t - nu).^2) ;    % A  target "PDF" ... Gaussian.
```

See the images below.

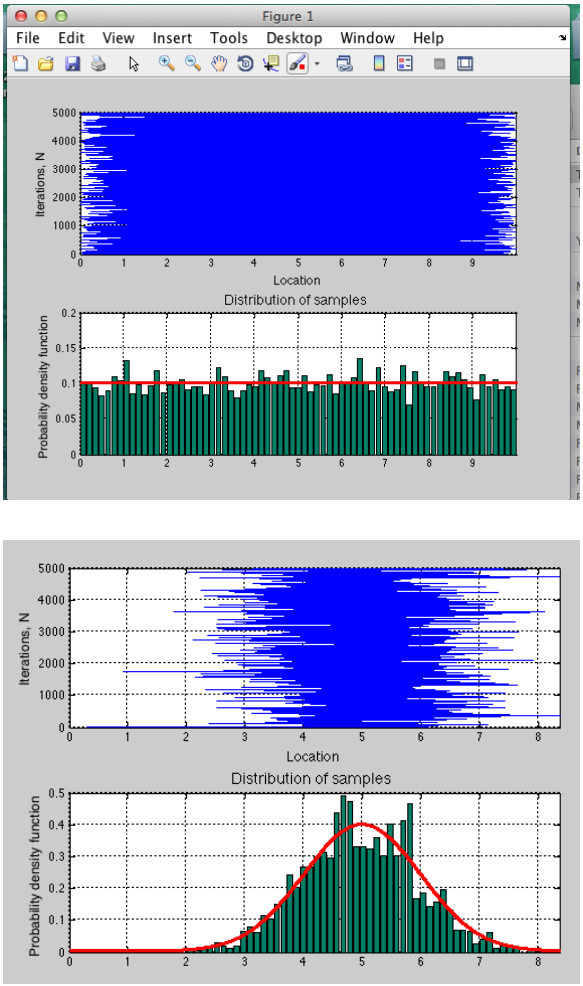

Figure 2: $N = 5000$ steps of the MMC accept/reject algorithm where we generate uniform random numbers between 0 and 10 or Gaussian random numbers with mean 5 and width 1. We can see both the values of the random variables (top part of each figure, in blue) and a histogram(bottom part of figure, green boxes) with the ideal probability distribution shown as a red curve.

It is also possible, as mentioned in the Intro, to use other criteria for choosing a new trial state, and for accepting the choice. Here is a different choice in lieu of Step. 5 in the methods section, that obeys detailed balance, Eq. (1):
Probability of acceptance: $\frac{\alpha}{1+\alpha}$
Probability of rejection: $\frac{1}{1+\alpha}$
where $\alpha = p(new\ state)/p(old\ state)$. In our code, the decision would be implemented as ...

```
alpha = p(theta_ast)/p(theta(i));
   if rand <= alpha / (1 + alpha)
    % Accept the sample with prob = alpha/(1+alpha)
   theta(i+1) = theta_ast;
else
   % Reject the sample with prob = 1/(1+alpha)
   theta(i+1) = theta(i);
```