Lecture 17: Monte Carlo Implementation

CBE 206

11.19.2019

Clarifications from Lecture 16

- In Lecture 16, we have encountered a problem of dense gas molecules inside a box.
- We wanted to compute the potential energy.
- We have stated that the following form is the correct one.

$$\langle U \rangle = \sum_{i_N}^{M} \dots \sum_{i_2}^{M} \sum_{i_1}^{M} U\left(r_{1,i_1}, r_{2,i_2}, \dots r_{N,i_N}\right) p\left(r_{1,i_1}, r_{2,i_2}, \dots r_{N,i_N}\right)$$

- This form above entails that we have M^N combinations on putting the molecules and is going through each point in a structured/systematic way.
- It is similar to doing Newton-Cotes integration in high dimensions.

Lecture 15: Newton Cotes → Monte Carlo

- We have learned from Lecture 15 that high dimensional integrals are best used using Monte Carlo.
- So instead of putting molecules in a structured, deterministic positions, we will randomly put them. Then, the expression changes (where we have used the Boltzmann factor as the probability).

$$\langle U \rangle \approx \frac{1}{Z} \sum_{i=1}^{N_{MC}} U\left(r_{1,i}, r_{2,i}, \ldots r_{N,i}\right) \exp\left[-\frac{U\left(r_{1,i}, r_{2,i}, \ldots r_{N,i}\right)}{k_B T}\right]$$

- However, we will realize that if we actually run a MATLAB code here, the convergence will be very slow (akin to problem 2 in Lecture 16).
- And that is due to the following.

Lecture 16: Monte Carlo is slow.

- For dense gas, most of the random N molecules generated inside a box will lead to two or more gas molecules overlapping.
- Let's say that gas molecule 1 and 2 are overlapping a lot.
- $U_{12} \gg 1$.
- Given the nature of the shape of the Lennard-Jones potential, the $|U_{12}| \gg |U_{ij}|$
- As such, it is likely that the entire potential energy gets dominated by few high energy pair-wise terms and $U(r_{1,i}, r_{2,i}, \ldots r_{N,i}) \gg 1$ as a result.
- This means that when we are doing Monte Carlo calculations, $\exp\left[-\frac{U(r_{1,i},r_{2,i},\ldots r_{N,i})}{k_B T}\right] \sim 0$.

- Now, $U(r_{1,i}, r_{2,i}, \ldots r_{N,i})\exp\left[-\frac{U(r_{1,i},r_{2,i},\ldots r_{N,i})}{k_B T}\right]$ is a term where due to exponential growth (or decrease in this case), even though $U(r_{1,i}, r_{2,i}, \ldots r_{N,i}) \gg 1$ the exponential dominates so $U(r_{1,i}, r_{2,i}, \ldots r_{N,i})\exp\left[-\frac{U(r_{1,i},r_{2,i},\ldots r_{N,i})}{k_B T}\right] \sim 0$

Lecture 16: Monte Carlo is slow.

- Putting all of this together, doing the Monte Carlo summation leads to the following for most randomly generated molecule configurations.

$$\langle U \rangle \approx \frac{1}{Z} \sum_{i=1}^{N_{MC}} U\left(r_{1,i}, r_{2,i}, \ldots r_{N,i}\right) \exp\left[-\frac{U\left(r_{1,i}, r_{2,i}, \ldots r_{N,i}\right)}{k_B T}\right]$$

$$\langle U \rangle \approx \frac{1}{Z}(0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + \cdots)$$

- And similar to problem 2 (Lecture 16), this “function” is peaked at a narrow range with most of the range yielding zero.
- This is bad..

Lecture 16: Metropolis-Hastings Algorithm

- We need a better algorithm to help us deal with these peaked function situations.
- The Metropolis-Hastings Algorithm helps us sample from an arbitrary probability distribution such that we are essentially doing "importance sampling" (sampling from region that are important and will yield non-zero values).
- There is a confusion about what it means by sampling from a probability distribution so an example can clarify.

Lecture 16: Metropolis-Hastings Algorithm and Sampling from Probability Distribution

- What does it mean when the algorithm is said to be sampling from the probability distribution?
- Let's say I have a normal dice and a weird dice with following probabilities.

Outcomes	Normal Dice (Probability)	Weird Dice (Probability)
1	1/6	1/12
2	1/6	1/12
3	1/6	1/12
4	1/6	1/12
5	1/6	2/12
6	1/6	6/12

Lecture 16: Metropolis-Hastings Algorithm and Sampling from Probability Distribution

- The average outcome is different between the normal and the weird dice.
- This is a simple problem that can be solved analytically, but I can also use Monte Carlo simulations. That is, this is the expression that I want.

$$\langle D \rangle = \sum_{i=1}^{6} D_i p_i$$

- I can write a simple MATLAB code that looks like the following..

```
Dtotal = 0;   % total sum of all outcomes of a dice
for k=1:N  % N (total number of Monte Carlo cycles)
       dice_number = some_probability_function(); (% returns
1 -> p1 %, 2 -> p2 %, .. 6 -> p6 %)
       Dtotal = Dtotal + dice_number;
end

Daverage = Dtotal / N  % average outcome
```

Lecture 16: Metropolis-Hastings Algorithm and Sampling from Probability Distribution

- For a normal dice, the probability of all outcomes are equal.
- So we can simply write a simple algorithm to randomly generate integer from 1, 2, 3, 4, 5, and 6.

```
Dtotal = 0;   % total sum of all outcomes of a dice
N = 1000000;
TotalOutcome = 6;
for k=1:N  % N (total number of Monte Carlo cycles)
       dice_number = randi(TotalOutcome);
       Dtotal = Dtotal + dice_number;
end

Daverage = Dtotal / N  % average outcome


>> Daverage = 3.5025  (% close to 21/6)
```

Lecture 16: Metropolis-Hastings Algorithm and Sampling from Probability Distribution

- For a weird dice, the probability of all outcomes are unequal.
- So we need to think a bit about how to write an algorithm to properly generate the correct probability distribution

```
Dtotal = 0;   % total sum of all outcomes of a dice
N = 1000000;
TotalOutcome = 6;
for k=1:N  % N (total number of Monte Carlo cycles)
       dice_number = randi(TotalOutcome);???????
       Dtotal = Dtotal + dice_number;
end

Daverage = Dtotal / N  % average outcome
```

Lecture 16: Metropolis-Hastings Algorithm and Sampling from Probability Distribution

- For a weird dice, we can try to construct some algorithm but this is quite a hassle.
- Weird dice algorithm.

```
r = rand();  % random number from 0 to 1
if (r < 1/12)
   dice_number = 1;
else if (r < 1/12 + 1/12)
   dice_number = 2;
…
```

- In general, we want a single, universal algorithm that can be applied to any probability distribution.
- That is the Metropolis-Hastings Algorithm.

Lecture 16: Metropolis-Hastings Algorithm and Sampling from Probability Distribution

- If we can sample from a probability distribution, then the expression for average of some quantity changes.

$$\langle A \rangle = \sum_{i=1}^{N} A_i p_i$$

$$\langle A \rangle = \frac{1}{N} \sum_{i=1}^{N} A_i$$

See pg. 7 for demonstration that p_i simply disappears. It becomes "implicit" in how we generate random numbers.

Lecture 16: Metropolis-Hastings Algorithm and Sampling from Probability Distribution

- Thus, for the average potential energy expression, p_i disappears as well.

$$\langle U \rangle = \frac{1}{N^M} \sum_{i_N}^{M} \dots \sum_{i_2}^{M} \sum_{i_1}^{M} U\left(r_{i_1}, r_{i_2}, \dots r_{i_N}\right)$$

Problem Type	Algorithm to Use (essentially getting rid of probability term)	Alternative Algorithm that you can Use
1. Normal Dice	randi (from page. 7)	Metropolis Algorithm
2. Weird Dice	If/elseif algorithm (from page. 9)	Metropolis Algorithm
3. Average Potential Energy	Metropolis Algorithm	Metropolis Algorithm

Lecture 16: Don't I Still Need to Evaluate Probability for Metropolis-Algorithm?

- Earlier on (page 3), we stated that evaluation of the Boltzmann probability is difficult because of evaluation of Z.
- But if we look at Slide 28 step 4), we see that we still need to evaluate the probability for the Metropolis Hastings algorithm.

- The beauty of the Metropolis-Hastings algorithm is that we do not need to evaluate the probability. We just need to evaluate the function $f(x)$ that is proportional to the probability. That is,

$$p(x) \rightarrow not\ needed$$

$$f(x) = \frac{p_{new}(x)}{p_{old}(x)} \rightarrow needed$$

- So then, we never need to evaluate Z as it cancels out in evaluating $f(x)$.

Lecture 16: Two Main Benefits of Metropolis-Hastings Algorithm

- 1) We can solve problems where direct probability distribution sampling is difficult (since we only need proportional probability sampling). Example: Z evaluation difficult)

- 2) Even if probability sampling is easy (i.e. $p(r)$ does not take too long to evaluate), Metropolis-Hastings algorithm is helpful when $p(r) = 0$ for most of the configuration r. The algorithm allows us to just sample in the "important" (i.e. high probability) region and thus, we do not waste our time.

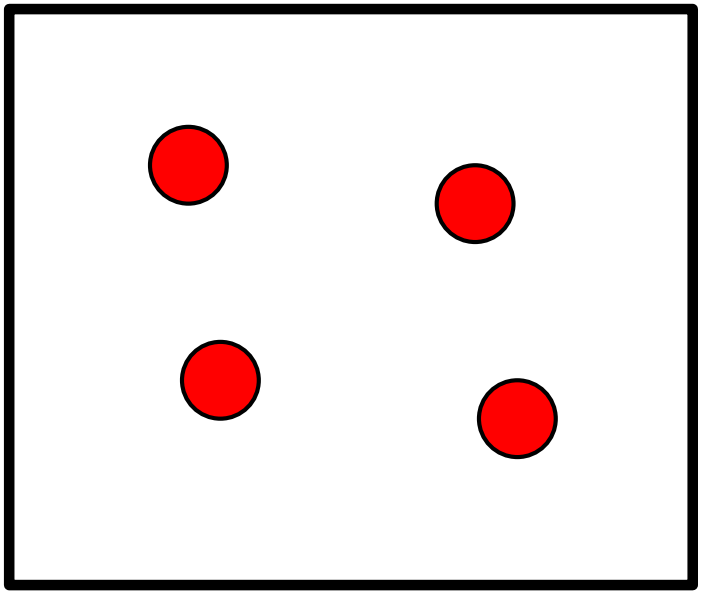
Motivation

- In Lectures 15 and 16, we have learned concepts behind the Monte Carlo algorithm and why they are used for certain situation. To summarize the various methods for integration thus far.
- Newton-Cotes method: use to solve low-dimensional integrals.
- Monte Carlo method:
  - Case 1: function is evenly distributed throughout the entire domain (use brute force method - Lecture 15.
  - Case 2: function is unevenly distributed and especially concentrated in a small parts of the domain (use selectively sampling such as Metropolis-Hastings algorithm to generate the distribution during the Monte Carlo iterations – Lecture 16).

- We will end the Lecture series on the Monte Carlo method with detailed MATLAB codes that you can use/modify for your benefit.
- The hope also is that going over the specific code, your understanding of the Monte Carlo algorithm will deepen throughout the process.

Sample System: Four Gas Molecules Inside a Box

- Let us take the example from Lecture 16 and simplify the system with four gas molecules inside the box.
- What is the average potential energy of the four gas molecules?
- $U_{ij} = 4\epsilon \left[\frac{\sigma^{12}}{r_{ij}{}^{12}} - \frac{\sigma^{6}}{r_{ij}{}^{6}} \right]$
- $\epsilon = 148.0\ K$ and $\sigma = 3.73$ Å.
- The temperature: $T = 300K$.
- Volume Size: $20 \times 20 \times 20$ Å^3.



- Let us use the Metropolis-Hastings algorithm (step-by-step).

Metropolis-Hastings Algorithm – Step 1

- Review from last lecture.
- Algorithm in a nutshell

1) Start with random configuration of N molecules
2) Select a molecule at random, and compute its energy, $U(R_{old})$.
3) Attempt to move the molecule to a new random position, $R_{new} = R_{old} + \Delta$, Δ is a random displacement. Compute its energy, $U(R_{new})$.
4) Accept or reject the move with probability, $\min\left(1, \frac{P(R_{new})}{P(R_{old})}\right) = \min\left(1, \exp\left[-\frac{U(R_{new}) - U(R_{old})}{k_B T}\right]\right)$.

Monte Carlo Code – MATLAB (Initialization)

```
% Metropolis-Hastings Monte Carlo Algorithm
NumMolecules = 4;   % total number of molecules
N = 100000;  % total number of Monte Carlo iterations
Epsilon = 148.0;  % parameter 1
Sigma = 3.73;  % parameter 2
Temp = 300;  % temperature in Kelvin (let it be 300K for now)
kB = 1;  % normalize to 1

% 1. Start with random configuration of molecules
Lx = 20.0;  % box length along x-direction
Ly = 20.0;  % box length along y-direction
Lz = 20.0;  % box length along z-direction

for k=1:NumMolecules
   x(k) = Lx*rand; % put randomly from 0 to Lx
   y(k) = Ly*rand; % put randomly from 0 to Ly
   z(k) = Lz*rand; % put randomly from 0 to Lz
end
```

Monte Carlo Code – Compute Initial Potential Energy

```
% 2. Compute initial potential Energy
UTotal = 0;   % store total potential energy in UTotal
for k=1:NumMolecules
   for kk=k+1:NumMolecules  % avoid double counting
      deltaX = x(k) - x(kk);
      deltaY = y(k) - y(kk);
      deltaZ = z(k) - z(kk);
      r = sqrt(deltaX^2 + deltaY^2 + deltaZ^2); % distance
      UPair = 4*Epsilon*(Sigma^12/r^12 - Sigma^6/r^6);
      UTotal = UTotal + UPair;  % update energy
   end
end
```

- If we started the index of the 2nd for loop from kk=1 (instead of k+1), UTotal would be two times what we obtain from above (since each interaction will be counted twice).

Metropolis-Hastings Algorithm – Step 2

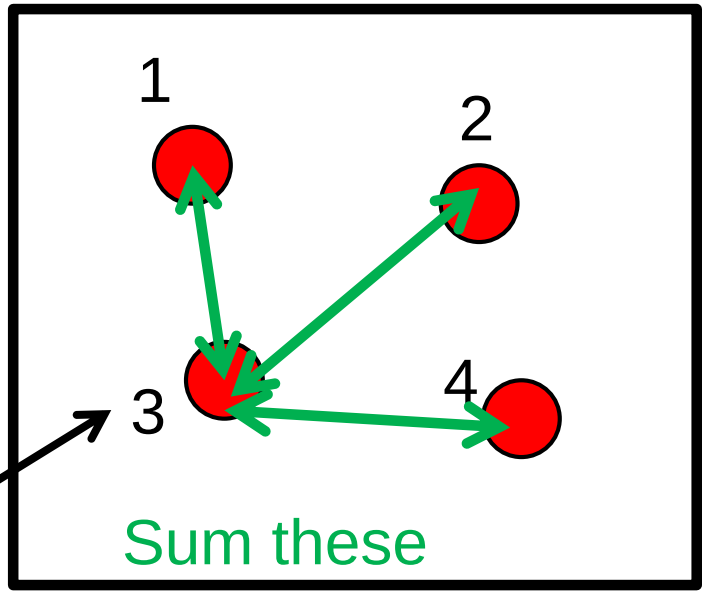
1) Start with random configuration of N molecules
2) Select a molecule at random, and compute its energy, $U(R_{old})$.
3) Attempt to move the molecule to a new random position, $R_{new} = R_{old} + \Delta$, Δ is a random displacement. Compute its energy, $U(R_{new})$.
4) Accept or reject the move with probability, $\min\left(1, \frac{P(R_{new})}{P(R_{old})}\right) = \min\left(1, \exp\left[-\frac{U(R_{new}) - U(R_{old})}{k_B T}\right]\right)$.

All of the moves here will be repeated N number of times inside the Monte Carlo loop.

Selecting a Molecule at Random and Computing Its Energy

- In general, each molecule has equal probability of being selected.
- In this example of four molecules, $p = 0.25$.
- The potential energy of a molecule (in this context) is different from the total potential energy. Let's say that we selected molecule $i = 3$ at random.

$$U_i = \sum_{j \neq i}^{NumMolecules} U_{ij}$$



Monte Carlo Code – Main Monte Carlo Loop

```
% 3. Monte Carlo loop
for k=1:N
   % 3a. Select a molecule at random
   % randi([1, NumMolecules]) generates random integer from 1 to
NumMolecules
   RandIndex = randi([1, NumMolecules]);

   % 3b. Compute its energy
   UOld = 0;
   for kk=1:NumMolecules
      if (kk ~= RandIndex)  % do not include self-interaction
         deltaX = x(RandIndex) - x(kk);
         deltaY = y(RandIndex) - y(kk);
         deltaZ = z(RandIndex) - z(kk);
          r = sqrt(deltaX^2 + deltaY^2 + deltaZ^2); % distance
          UPair = 4*Epsilon*(Sigma^12/r^12 - Sigma^6/r^6);
          UOld = UOld + UPair;  % update energy
      end
   end
   …
```

Metropolis-Hastings Algorithm – Step 3

1) Start with random configuration of N molecules
2) Select a molecule at random, and compute its energy, $U(R_{old})$.
3) Attempt to move the molecule to a new random position, $R_{new} = R_{old} + \Delta$, Δ is a random displacement. Compute its energy, $U(R_{new})$.
4) Accept or reject the move with probability, $\min\left(1, \frac{P(R_{new})}{P(R_{old})}\right) = \min\left(1, \exp\left[-\frac{U(R_{new}) - U(R_{old})}{k_B T}\right]\right)$.

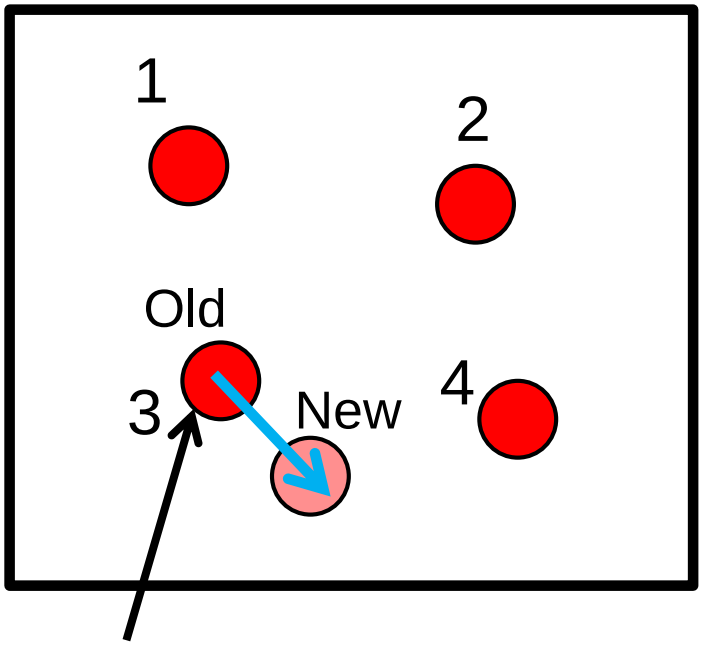
Attempt to Move the Selected Molecule at Random Displacement

- Generate three random numbers, $\Delta_x, \Delta_y, \Delta_z$.
- Move selected molecule from old to new positions.
- In general, you select the range of $\Delta_x, \Delta_y, \Delta_z$, such that there is not a bias towards any particular direction.

- E.g. Let the maximum possible displacement along each direction to be 1.0 Angstrom:

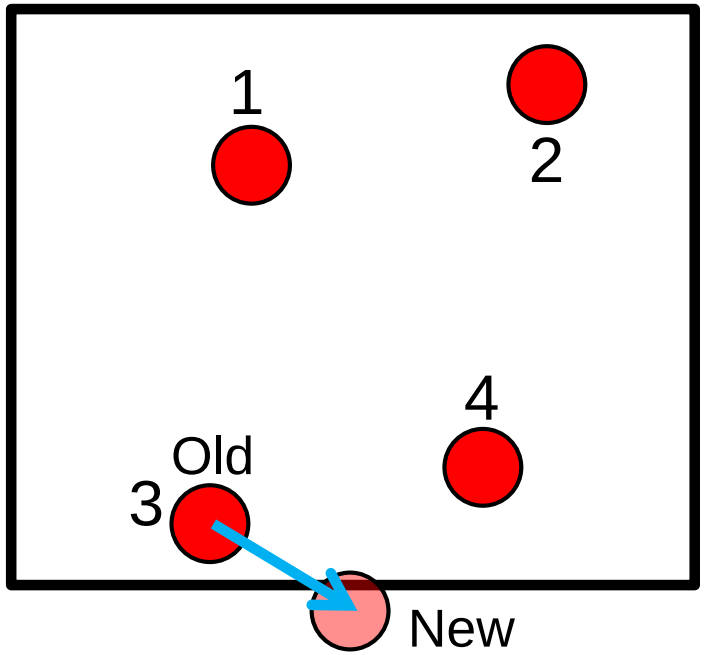
$\Delta_x = [-1, 1]$ $\Delta_y = [-1, 1]$ $\Delta_z = [-1, 1]$

$$x_{new} = x_{old} + \Delta_x$$
$$y_{new} = y_{old} + \Delta_y$$
$$z_{new} = z_{old} + \Delta_z$$



What Happens When We Attempt to Move the Molecule Outside the Box?

- For this problem, we assume that the molecules are confined inside the box: i.e. $U(R_{outside}) = \infty$.
- We will still attempt to move outside the box. But this attempt will always be rejected.
- So if there is a point anywhere in the code such that a molecule is outside the box, there is an error in the code.



Monte Carlo Code – Main Monte Carlo Loop (Continued from Slide 9)

```
…
% 3c. Move a molecule at random
xnew = x(RandIndex) + (2.0*rand - 1.0);
ynew = y(RandIndex) + (2.0*rand - 1.0);
znew = z(RandIndex) + (2.0*rand - 1.0);

% 3d. Compute the new energy
UNew = 0;
for kk=1:NumMolecules
   if (kk ~= RandIndex)  % do not include self-interaction
      deltaX = xnew - x(kk);
      deltaY = ynew - y(kk);
      deltaZ = znew - z(kk);
       r = sqrt(deltaX^2 + deltaY^2 + deltaZ^2); % distance
       UPair = 4*Epsilon*(Sigma^12/r^12 - Sigma^6/r^6);
       UNew = UNew + UPair;  % update energy
   end
end
…
```

Monte Carlo Code – Difference Between Old and New Energies

```
% 3d. Compute the old energy
UOld = 0;
for kk=1:NumMolecules
   if (kk ~= RandIndex)
      deltaX = x(RandIndex) - x(kk);
      deltaY = y(RandIndex) - y(kk);
      deltaZ = z(RandIndex) - z(kk);
      r = sqrt(deltaX^2 + deltaY^2 + deltaZ^2);
      UPair = 4*Epsilon*(Sigma^12/r^12 - Sigma^6/r^6);
      UOld = UOld + UPair;
   end
end
```

```
% 3d. Compute the new energy
UNew = 0;
for kk=1:NumMolecules
   if (kk ~= RandIndex)
      deltaX = xnew - x(kk);
      deltaY = ynew - y(kk);
      deltaZ = znew - z(kk);
      r = sqrt(deltaX^2 + deltaY^2 + deltaZ^2);
      UPair = 4*Epsilon*(Sigma^12/r^12 - Sigma^6/r^6);
      UNew = UNew + UPair;
   end
end
```

Metropolis-Hastings Algorithm – Step 4

1) Start with random configuration of N molecules
2) Select a molecule at random, and compute its energy, $U(R_{old})$.
3) Attempt to move the molecule to a new random position, $R_{new} = R_{old} + \Delta$, Δ is a random displacement. Compute its energy, $U(R_{new})$.
4) Accept or reject the move with probability, $\min\left(1, \frac{P(R_{new})}{P(R_{old})}\right) = \min\left(1, \exp\left[-\frac{U(R_{new})-U(R_{old})}{k_B T}\right]\right)$.

The above expression implies that we are taking the minimum between 1 and $\exp\left[-\frac{U(R_{new})-U(R_{old})}{k_B T}\right]$.

Accept/Rejection Criterion

- If $U(R_{new}) \leq U(R_{old})$, $\min\left(1, \exp\left[-\frac{U(R_{new})-U(R_{old})}{k_B T}\right]\right) = 1$
- That is, we always accept moves toward lower energies.

- If $U(R_{new}) > U(R_{old})$, $\min\left(1, \exp\left[-\frac{U(R_{new})-U(R_{old})}{k_B T}\right]\right) = \exp\left[-\frac{U(R_{new})-U(R_{old})}{k_B T}\right] < 1.$
- So there are nonzero chance of accepting moves toward higher energies.
- As $U(R_{new}) \gg U(R_{old})$, the likelihood of accepting the move decreases.
- We will randomly decide whether to accept/reject moves to higher energy.
- How to do this? Choose a random number, $r = [0,1]$.
- If $r < \exp\left[-\frac{U(R_{new})-U(R_{old})}{k_B T}\right]$, accept the move. If not, reject the move.

- If we accept the move, we will need to update x, y, and z vectors.
- If we accept the move, we will need to update the total potential energy, UTotal (from Slide 6).

Monte Carlo Code – Main Monte Carlo Loop (Continued from Slide 13)

```
    …
    % 3e. Metropolis-Hastings algorithm
    Metropolis = min(1, exp(-(UNew – UOld)/(kB*Temp)));
    MetroRand = rand;  % Select random number

    % first we need to reject any move outside the box
    if (xnew < 0 || xnew > Lx || ynew < 0 || ynew > Ly || znew < 0 ||
znew > Lz)
            % do nothing
    elseif (MetroRand <= Metropolis) % if it is inside the box and we
accept the move
        x(RandIndex) = xnew;  % update positions
        y(RandIndex) = ynew;
        z(RandIndex) = znew;
        UTotal = UTotal + (UNew – UOld);  % think about why this
expression is correct
    end
    …
```

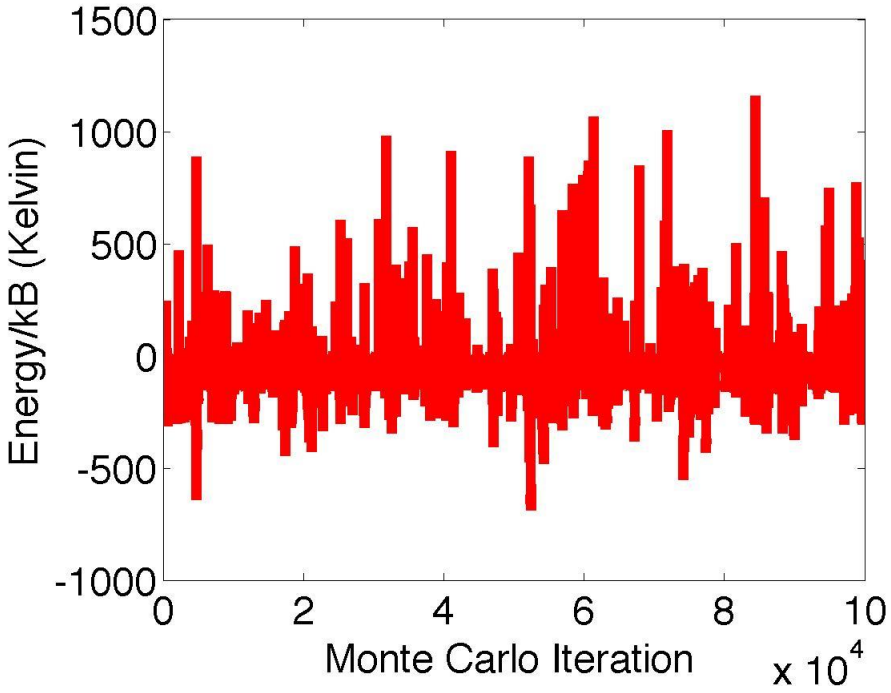
Monte Carlo Code – End of the Code (Continued from Slide 17)

```
    …
    % 3f.  Store the values of the current total energy inside an
array
    % We store the value regardless of whether or not the move has
been accepted or not.  There are still important information knowing
that the energy remained the same for a long time.
    EnergyArray(k) = UTotal;
    MonteCarloIterations(k) = k;
end  % End of Monte Carlo loop

% We can plot the Energy as a function of Monte Carlo cycles since
they have the same array size.
plot(MonteCarloIterations, EnergyArray, 'r-');

% End of code
```

Monte Carlo Code – Sample Run for Four Molecules



Computing Average Energy

- In general, to compute the average energy, we “throw away” the samples that were generated in the beginning of the Monte Carlo simulation.
- Why do we throw away samples?? Answer = the randomized initial configuration does not represent the “true” configuration that the system likes to be in at. This is an unfortunate by-product of the Metropolis-Hastings algorithm.

- How many samples should we throw away? Answer = this is a difficult question. But for our purpose, we will throw away 10% of the total number of Monte Carlo iterations. Thus, average energy looks as follows.

$$\langle U \rangle = \sum_{k=0.1N+1}^{N} \frac{U(k)}{0.9N}$$

Four Molecules, Monte Carlo Simulation Data Average Energy

Simulation	Average Energy/kB (Kelvin)
1	-40.4990
2	-50.8688
3	-44.5949
4	-39.3376
5	-42.5703
Total	-43.57 +/- 4.55

Monte Carlo Code – Sample Run for Four Molecules (Near Zero Kelvin)

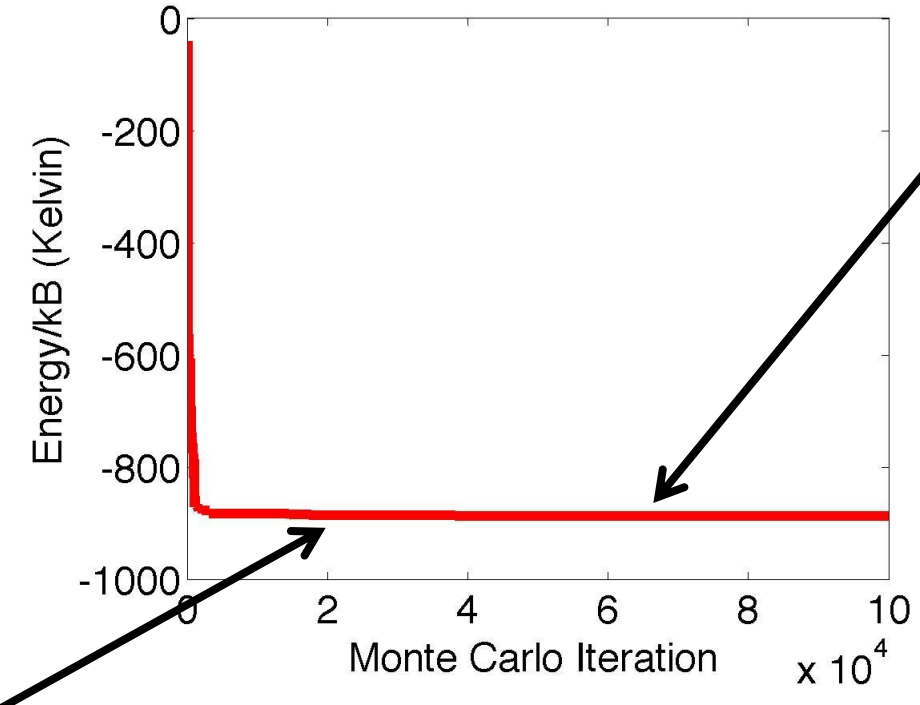
- Let us see what happens when we change the temperature from $T = 300K$ to close to zero. As an example, let $T = 10^{-10}K$.
- Nothing really changes in the code until the Metropolis/Hastings acceptance/rejection criterion, which is function of temperature.
- For small T

$$\min\left(1, \exp\left[-\frac{U(R_{new}) - U(R_{old})}{k_B T}\right]\right) = 1 \;\; when \;\; U(R_{new}) \leq U(R_{old})$$

$$\min\left(1, \exp\left[-\frac{U(R_{new}) - U(R_{old})}{k_B T}\right]\right) \approx 0 \;\; when \;\; U(R_{new}) > U(R_{old})$$

- Thus, under this extreme circumstance, we always accept moves to lower energies and reject moves to higher energies.

Monte Carlo Code – Sample Run for Four Molecules ($T = 10^{-10}K$)



$Energy = -887.51K$

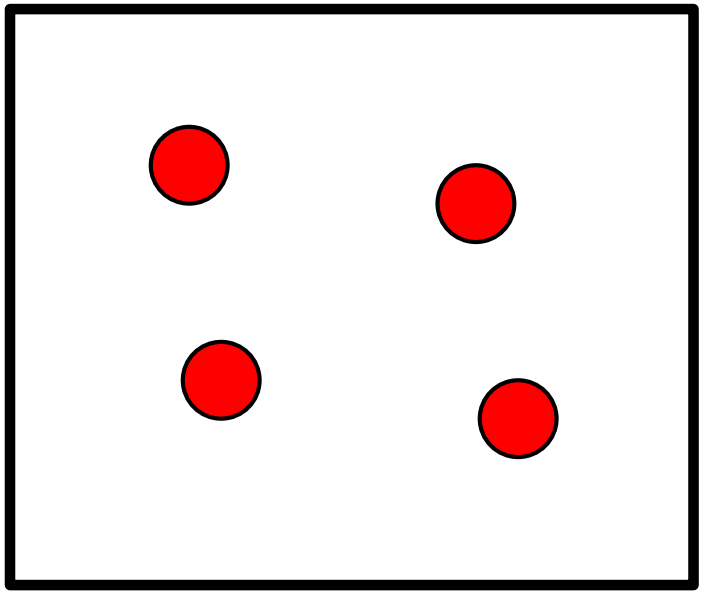
This value is closest to the global minimum energy for the four molecules configuration

Stabilizing at this value (different from T = 300K).
Stable because we cannot move to higher energies.

Global Minimum Energy for Four Molecules

- $U_{ij} = 4\epsilon\left[\frac{\sigma^{12}}{{r_{ij}}^{12}} - \frac{\sigma^{6}}{{r_{ij}}^{6}}\right]$
- $\epsilon = 148.0\ K$ and $\sigma = 3.73$ Å.
- $U_{ij,min} = -\epsilon = -148.0K$ (can be obtained by taking derivatives with respect to r)

- For four molecules, there are six pair-wise interactions ($U_{12}, U_{13}, U_{14}, U_{23}, U_{24}, U_{34}$).
- The molecules can be positioned such that all of these interactions are optimized.
- In that case, the global minimum energy is $U_{min} = 6U_{ij,min} = 6 \times -148.0 = -888.0K$
- We obtain an energy that is very close to this minimum value.



Summary

- In this lecture, we provided snippets of MATLAB codes to perform the Metropolis-Hastings algorithm Monte Carlo average potential energy calculations.
- The details here can help you understand the algorithm in better detail and implement your own Monte Carlo code for not only this problem but similar problems.