CHAPTER 4 - SYSTEMS AT FINITE TEMPERATURE

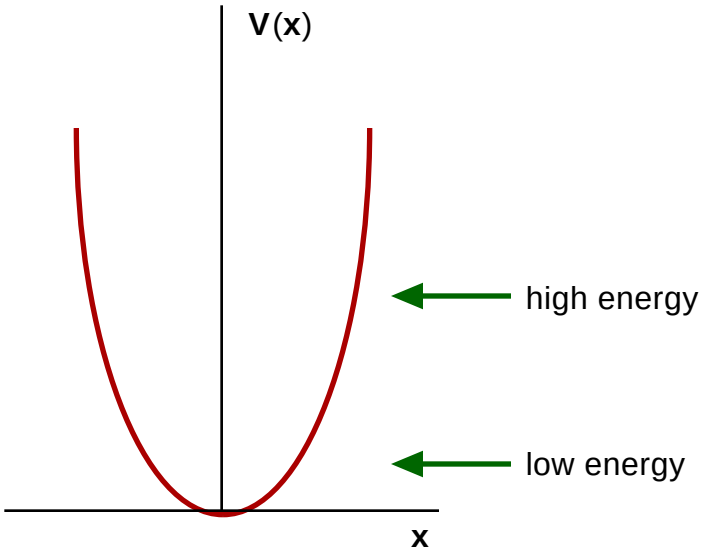
In isolation, a single object like a chain has a fixed energy. When placed in thermal contact with another system, the object's energy fluctuates as it exchanges energy with the system. In other words, the configurations that the object adopts do not all have the same energy. The question that we address in this section is: what is the probability that an object will have a given configuration with a specified energy? The answer to this question involves the concepts of *temperature* and a statistical factor called the *Boltzmann weight*, which is described and derived in Sec. 4.1 - 4.3.

The Boltzmann weight is used in several types of computational algorithms and is important for more than just problems in statistical mechanics. In Secs. 4.4 - 4.6, we describe one procedure for implementing the Boltzmann weight, called the Metropolis Monte Carlo algorithm. The Monte Carlo technique is used in the project of Sec. 4.7 to investigate the collapse of polymers with self-attraction, and is used in several other sections of this course, including non-ideal gases, simulated annealing, and phase transitions of spin systems.

4.1 Boltzmann weight

The intervertex potential of the linear chain in Chap. 3 is a step-function: it has hard walls at $x = a$ and $x = \sqrt{2}a$, and is constant for intermediate values of x, where x is the intervertex separation. Thus, all of the configurations that are allowed by the intervertex potential have the same potential energy, or equivalently, no configuration is energetically favored over another. This implies that the spatial configurations assumed by a chain whose vertices have high kinetic energy are equally accessible to a chain with low kinetic energy. Further, since all configurations have the same potential energy, *there is no unique ground state*.

This situation is different from many of the systems we examine in first or second year physics courses. Consider the motion of particle in a quadratic potential centered at $x = 0$:



At high total energy (kinetic plus potential), the particle can move through a modest range of x, while at low total energy, its range is more restricted. At zero energy, the particle occupies a unique ground state at $x = 0$. If this particle is placed in thermal contact with a large system at a fixed temperature, then we would find:

- high temperature large fluctuations in x
- low temperature small fluctuations in x
- zero temperature no fluctuations in x (ground state at $x = 0$).

In this potential, the configurations (*i.e.*, values of x) have differing potential energies depending upon the displacement from equilibrium. Thus, some configurations are accessible only to particles with high total energy. At a specified temperature, the particle is more likely to be found in some configurations than in others.

What is the probability for the particle to have a given value of x in the quadratic potential? As we show in Sec. 4.2, the probability $P(E)$ for a particle to have a specific configuration with energy E is

$$P(E) \quad \exp(-\beta E) \tag{4.1}$$

where

$$\beta = 1 / k_B T. \tag{4.2}$$

In Eq. (4.2), T is the absolute temperature and k_B is Boltzmann's constant, which has the physical value 1.38 x 10^{-23} J/K.

Note!
•Eq. (4.1) involves a proportional sign: the probability is not properly normalized by this equation.
•If several configurations have the same E, then *each configuration* has the same probabilty of occurence $P(E)$.

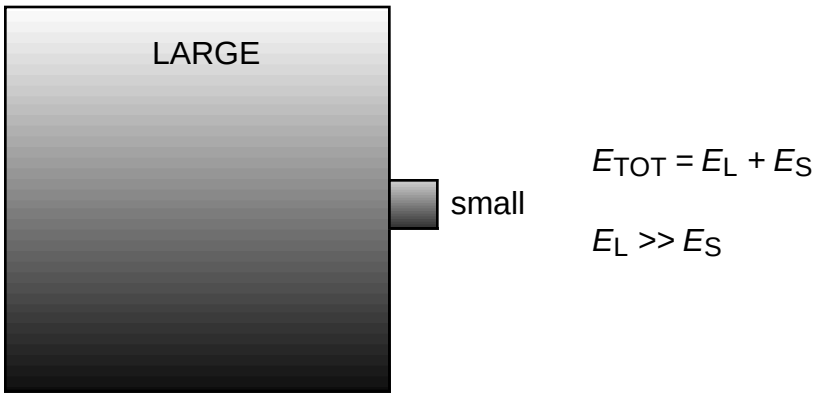
4.2 Derivation of Boltzmann factor

There are several conventional derivations of the Boltzmann factor. We present a compact derivation from Reif (1965), although our notation is slightly different from his.

Consider two systems in thermal contact with each other, but otherwise isolated. The two systems are not equal "size" in the sense that one system is assumed to have far more energy than the other. The system with the large amount of energy is denoted by L, and the system with the small amount of energy is denoted by S. The total energy of the two systems, E_{TOT} is a constant:

$$E_{TOT} = E_L + E_S. \qquad (4.3)$$

The two systems can exchange energy, but a given change in energy ΔE results in only a very small *relative* change in the energy of the large system.



Consider now the situation where the total energy of the two systems is fixed at some value E_o; that is, we now specify E_{TOT} at some numerical value, say $E_{TOT} = E_o = 10^6$ Joules. We are interested in a particular state *r* of the small system. This state has energy E_r,

$$E_S = E_r \tag{4.4}$$

while the large system has energy

$$E_L = E_o - E_r. \tag{4.5}$$

With E_{TOT} fixed at E_o, the probability that the small system will be in the single state *r* is proportional to the number of states in the large system with energy $E_o - E_r$. That is, if there are few states in the large system with the specific value of E_L given by Eq. (4.5), then the probability of the small system being in the specific state *r* will be low.

We denote the number of states of the large system having energy E_L to be

$$\Omega(E_L) = [\textit{number of states of large system with } E_L]. \tag{4.6}$$

If the probability of the small system being in state *r* is P_r, then

$$P_r = A\, \Omega(E_L), \tag{4.7}$$

where *A* is a constant that is a characteristic of the *large* system. The value of *A* can be determined through the condition that the small system must always occupy an available state, although it may change states as time passes:

$$\sum_r P_r = 1, \tag{4.8}$$

where the sum is over all of the states *r* available to the small system.

The number of states of the large system $\Omega(E_L)$ will vary with E_L, perhaps rapidly, depending on the value of E_L. The logarithm of $\Omega(E_L)$ varies much more slowly, and can be expanded in a series around $E_L = E_o$:

$$\ln \Omega(E_o - E_r) = \ln \Omega(E_o) - [\partial \ln \Omega / \partial E\,]_o\, E_r + ... \tag{4.9}$$

where the derivative in Eq. (4.9) is with respect to the energy of the *large* system. The minus sign in front of the derivative arises since the energy E_L of the large system decreases by E_r when the energy of the small system increases by E_r. Higher order terms can be neglected since $E_r << E_o$. Now, the derivative $[\partial \ln \Omega / \partial E]_o$ is a characteristic of the large system around $E_L = E_o$, and does not depend on E_r. We define the derivative as

$$[\partial \ln \Omega / \partial E]_o \equiv \beta. \qquad (4.10)$$

It can be shown that β^{-1}, which has the units of energy, has the properties of a temperature, and the absolute temperature scale is set through Boltzmann's constant

$$\beta^{-1} = k_B T, \qquad (4.11)$$

where $k_B = 1.38 \times 10^{-23}$ J/K.

Returning now to Eq. (4.9), the derivative can be replaced by the inverse temperature

$$\ln \Omega(E_o - E_r) = \ln \Omega(E_o) - \beta E_r + ...$$

$$\Omega(E_o - E_r) = \Omega(E_o) \exp(-\beta E_r). \qquad (4.12)$$

Finally, substituting back into the expression (4.7) for the probabiltity

$$P_r = [A\, \Omega(E_o)] \exp(-\beta E_r). \qquad (4.13)$$

This is Boltzmann factor, and it shows that the probability of the small system being in a specific state r with energy E_r is a function of the energy of the state and of the temperature of the large system with which it is in thermal contact. The two factors in the square braces of Eq. (4.13) are both constants, and can be rolled into one as

$$P_r = C \exp(-\beta E_r) \qquad (4.14)$$

with C determined from

$$C^{-1} = \sum_r \exp(-\beta E_r). \qquad (4.15)$$

4.3 Example: harmonic oscillator

As an application of the Boltzmann factor, we consider the motion of a particle in the quadratic potential

$$V(x) = k_{sp}x^2 / 2, \tag{4.16}$$

where k_{sp} is the spring constant and x is the displacement from equilibrium. For this potential, the constants in the exponential of Eq. (4.14) always occur in the combination $\beta k_{sp}/2$, so that it is convenient to replace them with a new constant α

$$\alpha \equiv \beta\, k_{sp} / 2. \tag{4.17}$$

Now x is a continuous variable, so one talks of the probability $P(x)dx$ of the particle having a displacement between x and $x + dx$. For continuous variables, Eqs. (4.14) and (4.15) become

$$P(x)dx = dx \bullet \exp(-\beta k_{sp}x^2 / 2) / \int \exp(-\beta k_{sp}x^2 / 2)\, dx$$

$$= dx \bullet \exp(-\alpha x^2) / \int \exp(-\alpha x^2)\, dx. \tag{4.18}$$

Eq. (4.18) can be used to evaluate the expectation value of x^2

$$\langle x^2 \rangle = \int x^2 P(x)\, dx$$

$$= \alpha^{-1} \int z^2 \exp(-z^2)\, dz \; / \int \exp(-z^2)\, dz$$

$$= \alpha^{-1} (\sqrt{\pi} / 2) / \sqrt{\pi} \; = 1 / 2\alpha \tag{4.20}$$

$$\langle x^2 \rangle = 1 / \beta k_{sp} = k_B T / k_{sp}. \tag{4.21}$$

Eq. (4.21) shows that $\langle x^2 \rangle$ increases with temperature T. At zero temperature, the fluctuations in x^2 go to zero, as the particle settles into its ground state at $x = 0$.

4.4 Metropolis algorithm

In Sec. 4.2, we established that the probability of a system having a specific configuration with energy E_r is proportional to the Boltzmann weight $\exp(-\beta E_r)$. This means that the relative probability of a system being in one of two specific states $r = 1$ and 2, separated in energy by $E_1 - E_2$ is

$$P_1 / P_2 = \exp[-\beta(E_1 - E_2)]. \qquad (4.22)$$

What computational algorithm will lead to an equilibrium distribution of the form (4.22)? There are several algorithms available, the simplest of which is referred to as the Metropolis algorithm (after its inventor), perhaps more commonly called the Monte Carlo algorithm.

In the Monte Carlo approach, one makes a trial move on a component or characteristic of the system, and then accepts or rejects the move according to a set of rules. For example, in the linear chain problem of Chap. 3, a trial move is made on a position of a vertex of the chain, and the move is accepted or rejected according to whether the new position violates the step-function potential between chain vertices. As we have stressed, there is no energy scale in the step-function potential of Chap. 3, so the move is accepted or rejected with unit probability. More generally, however, there is an energy change with the trial move, and the Metropolis algorithm assigns the probability of acceptance as

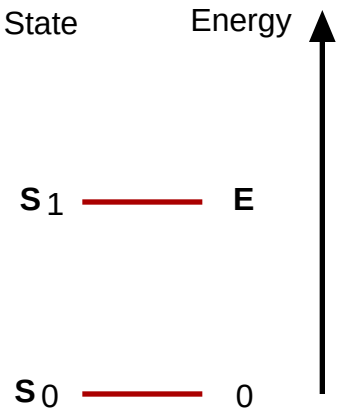
$$\begin{aligned} P &= \exp(-\beta\Delta E) \qquad \text{for } \Delta E > 0 \\ P &= 1 \qquad\qquad\quad \text{for } \Delta E \le 0, \end{aligned} \qquad (4.23)$$

where $\Delta E = E_{new} - E_{old}$ is the energy change associated with the trial move. The second of these equations is consistent with the first, in that P would exceed unity if the exponential is evaluated with $\Delta E \le 0$.

There are many subtleties, variations and pitfalls to the Monte Carlo technique, and several of these are illustrated in the following example.

4.5 Example: two-state model

As an illustration of the Monte Carlo procedure, consider the following system. Let there be two states to the system, S_0 and S_1: the state with lower energy provides the zero-energy reference point, while the other state has energy E.



The probabilities for the system to occupy the states are:

$$P_0 = A$$
$$P_1 = A\exp(-\beta E), \tag{4.24}$$

where A is a normalization constant determined through the condition

$$P_0 + P_1 = 1. \tag{4.25}$$

Hence,

$$P_0 = 1 / [1 + \exp(-\beta E)]$$
$$P_1 = \exp(-\beta E) / [1 + \exp(-\beta E)]. \tag{4.26}$$

For example, if the temperature is set to the specific value

$$\beta E = \ln 2, \tag{4.27}$$

then the probabilites are

$$P_0 = 2/3$$
$$P_1 = 1/3. \tag{4.28}$$

Knowing the relative probability for the occupation of the states, one can extract the ensemble-average of the energy $\langle E \rangle$

$$\langle E \rangle = P_0 E_0 + P_1 E_1 = (2/3)\bullet 0 + (1/3)\bullet \ln 2 / \beta \qquad (4.29)$$

$$\langle \beta E \rangle = (\ln 2) / 3. \qquad (4.30)$$

It is simple enough to write a code to simulate this system, using the probabilities from Eq. (4.23):

$$P(0 \rightarrow 1) = \exp(-\ln 2) = 1/2$$
$$P(1 \rightarrow 0) = 1. \qquad (4.31)$$

The following code generates an ensemble of states, and evaluates $\langle E \rangle$. The headers and random number generator are included for completeness. Eq. (4.30), or any finite-temperature variation of Eqs. (4.26), can be verified using this code.

```
#include <stdio.h>
#include <math.h>

/* Two-state Monte Carlo */

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define MASK 123459876

long seed;
float ran0(long *idum);

void main()
{
  int i, state, n0, n1;
  double beta, expmb, p0, p1, aven, theory;
  seed=234243;
  beta=log(2.0);     /* beta*energy of state 1 */
  expmb=exp(-beta);
```

```
  state=0; n0=n1=0;  /* initial state and counters */
  for(i=0; i<10000; i++) {
    if(state == 1) {state=0; n0++;}
    else{
      if(expmb > ran0(&seed)) {state=1; n1++;}
        else n0++;
    }
  }
  p0=0.00010*n0; p1=0.00010*n1; aven=0.00010*beta*n1;
  theory=beta*expmb/(1.0+expmb);
  printf("Probability of state 0 = %f\n",p0);
  printf("Probability of state 1 = %f\n",p1);
  printf("<beta*E> Monte Carlo = %f\n",aven);
  printf("<beta*E> theory = %f\n",theory);
}

float ran0(long *idum)
{
  long k;
  float ans;
  *idum ^= MASK;
  k=(*idum)/IQ;
  *idum = IA*(*idum-k*IQ)-IR*k;
  if(*idum < 0) *idum += IM;
  ans = AM*(*idum);
  *idum ^= MASK;
  return ans;
}
```

4.6 Subtleties and variations

The Metropolis algorithm of Eq. (4.23) is not the only way to sample configuration space. Further, the random moves in Cartesian coordinates that we use in Projects 3 and 4, while correct, may not be the most useful coordinates for some geometries. We turn to a few of these issues now.

Other acceptance algorithms

We have illustrated through example, and it can be shown analytically, that the Metropolis algorithm for accepting/rejecting trial moves generates an ensemble of

configurations that obeys the Boltzmann factor. However, there are other algorithms for accepting moves (Glauber, for example) that have slightly different forms from the Metropolis one. In general, these algorithms have slightly longer execution times.

Trial moves

There are two aspects to making a trial move. The first is the selection of the attempted move itself, and the second is the evaluation of whether the move is acceptable. Eq. (4.23) deals with the second step. In the linear chain projects, we tacitly and correctly assumed that the first step involved choosing a new x, y or z coordinate *uniformly* from a range Δs centered on the particle's current position.

Suppose that we wanted to sample trial moves from within a *spherical volume*, rather than a *cubical volume*, centered on the particle's current position? It is tempting to use polar coordinates r, θ and ϕ, rather than x, y and z for such moves. However, one must be careful in working in such coordinates to assign the correct weight to the move. Just as the infinitesimal volume element in polar coordinates is $r^2 dr\, d\cos\theta\, d\phi$, and not $dr\, d\theta\, d\phi$, so too, the sample must be uniformly drawn from:

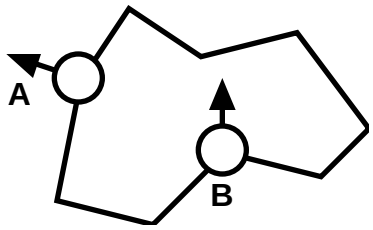
$$\Delta r^3 \qquad \Delta\cos\theta \qquad \Delta\phi.$$

If one does not select uniformly from $\cos\theta$, for example, then the trial moves of the particle are incorrectly biased towards the polar (*i.e., z*) direction.

Conserved moves

Consider the situation in which the linear chain closes upon itself to form a ring in two dimensions. Then a trial move on a single vertex not only changes the configuration, it also changes the enclosed area of the ring. While it may be acceptable to change the ring's area in some situations, it may be required to fix the area in other situations. There are two common ways to enforce a conservation law:

Strict enforcement by moving two particles/elements at once and forcing the move to conserve the quantity of interest:



For example, one could move particles **A** and **B** simultaneously in a way that conserved the enclosed area of the ring.

Approximate enforcement by adding a term to the energy that biases the quantity of interest towards a specific value. For example, the area could be approximately conserved by adding a term like $K\,(A - A_0)^2 / A_0$, where K is a user-supplied constant and A_0 is the "conserved" area. The larger the value of K, the closer the system's area will stay to A_0.

Ordered moves

We made no mention about the *order* in which the vertices were to be moved in the linear chain project: making trial moves sequentially from vertex 1 to vertex N appears to be acceptable. However, there are situations in which one wants to choose a vertex or element *randomly* from the list of objects to be moved, for example as

vertex 2 vertex 29 vertex 6 vertex 2 vertex 11 vertex 13

Note that it is not necessary to sweep through each and every vertex before starting the moves again: in the example, sequential calls to vertex 2 may, at times, be more or less frequent than average, as long as vertex 2 is not moved more frequently than average over long time periods.

References

F. Reif. 1965. *Fundamentals of Statistical Mechanics* (McGraw-Hill, New York). Chap. 6.

4.7 Project 4 - Chains with attraction

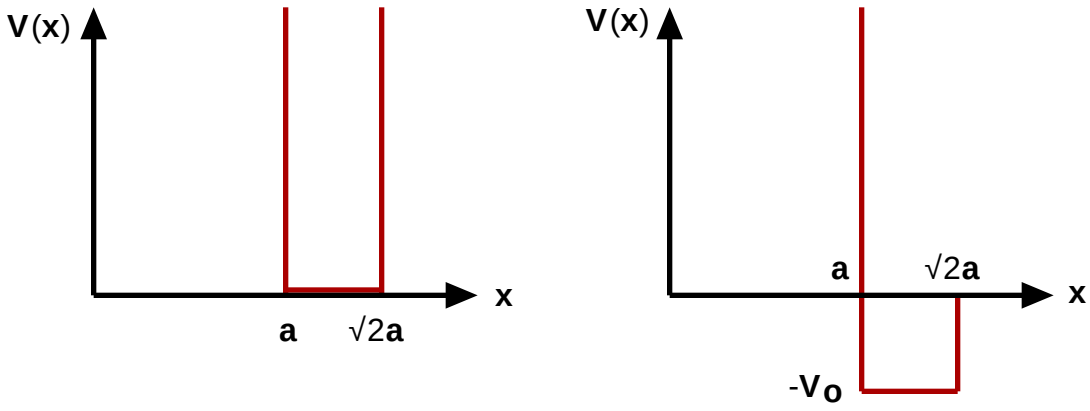
Project 4 builds upon the linear chain code written for Project 3. In addition to interactions between neighboring vertices along the chain, we now include interactions between non-neighboring vertices:

- •short-range repulsion to enforce self-avoidance
- •short range attraction.

Allowing non-neighboring vertices to interact will slow down the execution time of the code, in principle. There are of order N pairs of neighboring vertices on a chain, but of order N^2 pairs of non-neighboring vertices, where N is the number of vertices. One way to avoid the code grinding to a halt at large N is to make use of neighbor lists, which are explained in the section below on *Programming Hints*. **BUT DON'T START MODIFYING YOUR CODE UNTIL YOU'RE SURE THAT IT WORKS AND PRODUCES THE CORRECT PHYSICS FOR SOME TEST SYSTEMS.**

Code

In addition to the "tethering" potential between neighboring vertices used in Project 3, add a potential between all non-neighboring vertices:



Neighboring Non-neighboring

There is no energy or temperature scale in the tethering potential, since the potential energy is a fixed constant in any region of x where it is finite (would there be an energy scale if $V(x) = C$ for $a < x < \sqrt{2}a$ rather than $V(x)$=0?). However, the interaction between non-neighboring vertices has both a hard-core repulsion region for $x < a$, as well as a short-ranged attraction region for $a < x < \sqrt{2}a$. The choice of $\sqrt{2}a$ for the outer limit of the attractive region helps to speed-up the code (in that the interactions change at only two values of x: a and $\sqrt{2}a$), but is otherwise arbitrary. In the attractive region, $V(x) = -V_O$, and this provides an energy scale.

A short comment on the step-function potential. Many potentials for physical systems don't have the discontinuities associated with step-functions. Further, if we were interested in calculating forces by differentiating a potential, step-functions are not a good choice. However, the characteristics of a system, such as the specific heat, the presence of a phase transition, *etc*., are not particularly sensitive to the functional form of the interaction potential. For example, the $N^{1/2}$ scaling of the end-to-end displacement in a random walk is found for a variety of linear chain models, including:

- •fixed step length on a lattice
- •variable step length
- •step function interaction for neighboring vertices
- •smooth, differentiable functions for intervertex interactions
- •fixed polar angle between chain segments.

The scaling behavior is *universal*: it is the same for all chains that are *linear*, *flexible*, and *not-self-avoiding*. Of course, the prefactor C of the scaling law $\langle r_{ee}^2 \rangle = CN$, does depend upon the model, but the value of the exponent does not. Our philosophy in this course is to choose the simplest model that displays the correct physics. The idea that system behavior may be *universal* always should be kept in mind in designing simulation models.

The attractive interaction introduces an energy scale, and hence a temperature: $\beta V_o = V_o / k_B T$ is a dimensionless parameter of the system. One of the objectives of this project is to find the dependence of the system size on temperature, or more precisely, on βV_o. Because of the energy scale, a given trial move on a vertex position may:

(i) be rejected because it violates a bound (*e.g.*, $x < a$)

(ii) be accepted because there is no change in energy (*i.e.* $\Delta E = 0$ for the total change in energy of the system, considering all of its interactions)

(iii) be accepted conditionally if there is a change in energy [$\Delta (E / V_o) = \pm 1, \pm 2, \pm 3...$ depending on the number of particles within interaction range of a given particle].

Build your project on the code you wrote for project 3: linear chain with N vertices moving in three dimensions. To this, add

- •self-avoidance and attractive interactions between non-neighboring vertices
- •Metropolis algorithm for the acceptance criteria of energy-changing moves.

From Eq. (4.23), the probability of acceptance of a move is

$P = \exp(-\beta \Delta E)$ for $\Delta E > 0$

$P = 1$ for $\Delta E \leq 0$.

The implementation procedure for the conditional probability is to throw a random number and accept the move if the random number is less than $\exp(-\beta\Delta E)$.

Analysis

The end-to-end displacement is not particularly useful for characterizing the geometry of chains with attraction. Instead, use the radius of gyration, defined by $r_g^2 = N^{-1} \sum_i r_i^2$ (translate the center-of-mass position of your chain to the origin before using this definition!). After evaluating r_g^2 for each statistically independent configuration, determine the ensemble average $\langle r_g^2 \rangle$. Your analysis should include:

1. The power-law scaling of $\langle r_g^2 \rangle$ with N at $V_o / k_B T = 0.5$ (or less if the scaling isn't well-defined). How is the scaling exponent different from the ideal random walk? Check back to Sec. 3.4 on self-avoiding walks for the expected scaling exponent.

2. The power-law scaling of $\langle r_g^2 \rangle$ with N at $V_o / k_B T = 1.0$ (or more if the scaling isn't well-defined). How is the scaling exponent different from the ideal random walk, or the result found for $V_o / k_B T = 0.5$? The scaling behavior of dense systems is discussed at the end of Sec. 3.4.

3. An estimate of the uncertainties of your results.

Report

Your report should include the following items;
- paragraphs on the scaling expected for random, self-avoiding and dense chains
- an outline of your code
- your data for $\langle r_g^2 \rangle$
- analysis of the combined class data to obtain the scaling exponent
- an estimate of the uncertainties of your data and extracted exponents
- a copy of your code.

Demonstration

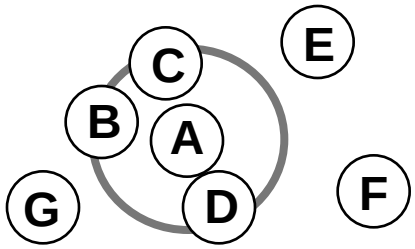
The demonstration code 395Proj4 illustrates the behavior of chains with attractive interactions. Choose a value for N, then run the code at each of the temperatures. How does the "size" of the chain change with temperature (temperature

decreases as $V_0 / k_B T$ increases)? The demonstration can be run in both 2 and 3 dimensions - does the dimensionality affect the temperature-dependence of the chain configurations?

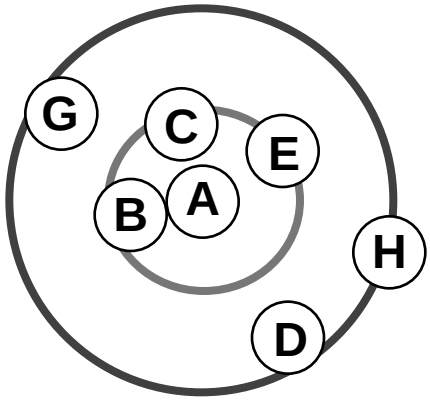
Programming hints: neighbor lists

As indicated above, interactions between non-neighboring vertices introduce elements to your code that scale like N^2, not good news for running code at large N. This turns out to be a generic property of interactions, so it is worth worrying about if you do scientific programming for a living. There is a literature on how to speed up simulations of interacting systems, but here we present only two variations on a theme. Wait until your code actually works correctly, before modifying it to increase its speed. **KEEP COPIES OF WORKING CODES!**

If the interaction between non-neighboring vertices is long range (as in a potential like $V(r) \sim 1/r$), then all particles affect all others. However, if the interaction is short-ranged, as in the step-function example, then only those particles whose separations are less than $\sqrt{2}a$ affect each other. Consider particle A in the diagram below. Ignoring the existence of tethering interactions for the moment, there are three particles (B, C, D) close enough to A in the diagram to experience an attractive interaction with A. The interaction region lies within the gray circle.



If we look at the environment around A a few trial moves later, there has been some movement of particles, with particle D no longer interacting with A, and E now within the range of interaction. Particle F has left the scene, while H has arrived nearby.



We can define a region, indicated by the outer circle, beyond which *NO* particle could have reached A within the elapsed number of trial moves. If the number of moves is small, then the outer circle is small; clearly the size of the circle grows with time.

The number of particles within the outer circle may be 5, 10, 30.. but it is not *N*. Thus if one knew which particles were within the outer circle at a given time, then one need only evaluate their contribution to the potential energy for the next several trial moves. One would not need to evaluate all *N* particles. Studies have been done on finding the size of the outer radius that optimizes the code execution speed. For our purposes, an effective strategy is

1. Evaluate the separation between all pairs; for each particle (*e.g.* A), store the labels of the other particles lying within 3*a* (*e.g.* B, C, D, E, F, G). This is the *neighbor list* of A.

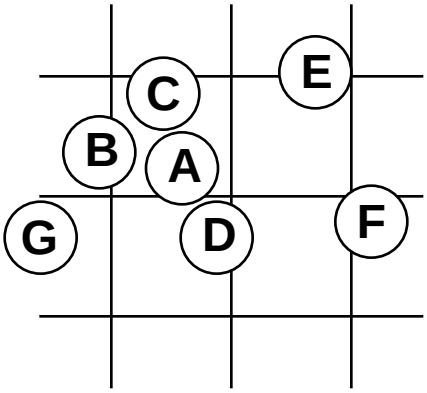
2. For the next ten steps per particle, evaluate the potential energy using only the particles in the neighbor list (B...G). Use a position step size *ds* / *a* = 1/10. If you use a larger position step size, then a larger neighborhood radius may be needed.

3. Update the neighbor lists of all particles after 10 steps.

4. Be aware of the machine memory limitations when you declare the neighbor lists. Since the purpose of the list is to reduce the evaluation of pairwise interactions, only a fraction of the system will be in the neighbor list of each particle. Thus, there is no need to declare an *N* x *N* array **list[N][N]** to store the neighbor lists. Something much more modest, typically **list[N][30]**, should suffice. Indeed, for some operating systems, arrays larger than 250x250 are not permitted. Of course, if the configuration is very dense, then there may be more than 30 particles within 3*a* of a given chain element.

Once you have a working code, you may want to play with the neighborhood radius to find its optimal value. For example, what is the maximum radius for the neighborhood, given by the condition that two particles move towards each other at the maximum individual step size each time? This radius is obviously overkill in a Monte Carlo routine, since the particles are not guided directly by forces, but take a less certain path, like Life itself.

An alternate approach to particle-based lists uses a set of neighborhoods that are fixed in space, and follows the motion of particles from one region to another. The situation is illustrated below



The straight lines indicate the fixed boundaries of the neighborhoods. The advantage to the fixed neighborhoods is that it is an order N operation to assign all the particles to boxes, rather than the N^2 evaluation of distances required to assign particles to moving neighborhoods. The disadvantage to the technique is the overhead required to update the lists of particles in each box, and the memory taken up by the fixed array.

A typical algorithm would involve the following steps:

•assign particles to boxes
•evaluate the Boltzmann weight for a trial move using interactions with all particles in the same box and in nearest-neighbor boxes
•update the particle lists for each box after accepting the moves.

The algorithm may become inefficient if the number of particles per box becomes too large. Similarly, if the boxes are small compared to the range of the interaction, then next-nearest-neighbor boxes (and perhaps more!) must be checked in evaluating the potential energy of a particle.