1 Molecular-dynamics algorithms

Discuss the algorithms for molecular-dynamics modeling:

- Potentials
- Integration
- Cut-off
- Periodic boundary Conditions
- Initialization
- Efficiency improvements

Potentials

The potential function is the most crucial part, as it is what defines the interaction between the particles. Some potentials are defined between pair of particles, so called two-body potentials. Sometimes we invoke three-body or many-body potentials.

A good example of a two-body potential is the Leonnard-Jones potential, aka the 6–12 potential.

$$U(r) = 4\epsilon\left[\left(\frac{r}{\sigma}\right)^{12} - \left(\frac{r}{\sigma}\right)^{6}\right].$$

The ϵ and σ are parameters that can be fitted to simulate different gases. In principle, the LJ-potential is used to simulate noble gases, as covalent and metallic bonding is more complex. The ϵ paramter gives the 'strength' of the force, and the σ gives the length scale. The 12-term describes Pauli repulsion, the 6-term describes dipole-dipole attraction (van der Waal-force).

Most practical potentials are specially constructed/fitted to be specific to a given problem/system. For example, the **Weber-Stillinger** potential is made to describe silicon. In equilibrium silicon aligns in tetrhedral structures with tight bonds. A spherical symmetrical potential cannot caputre this behaviour in a meaningful way. The W-S potential does it by combining a two-body and a three-body contribution. The three-body part describes the bedning stiffness of the system and the two-body part is L-J like.

Integration

It is very common to use the velocity-verlet method to integrate Newton's equations of motion. It is effectively a leap-frog method without a staggered grid, meaning the velocity equation is solved using the midpoint method in position and vice versa. In practice, this is done by computing the velocity in two half-steps. The velocity-verlet is therefore sometimes called half-kick/drift/half-kick.

So we have

$$\begin{aligned} v_{i+1/2} &= v_i + a_i \frac{\Delta t}{2}, \\ x_{i+1} &= x_i + v_{i+1/2}\Delta t, \\ v_{i+1} &= v_{i+1/2} + a_{i+1}\frac{\Delta t}{2}. \end{aligned}$$

Note that to perform the last step, the acceleration must first be evaluated from $a_{i+1} = -\mathrm{d}U(x_{i+1})/\mathrm{d}x$, which is why the velocity uses the midpoint method.

As for the leapfrog method, velocity-verlet is of global order $\mathcal{O}(\Delta^2)$ as it uses the midpoint method to solve the equations of motion per time step, which has the error $\mathcal{O}(\Delta t^3)$, but then we repeat this for $N = T/\Delta t$ steps.

The benefits of velocity-verlet are

- Preserves the time-reversability of the equations of motion
- Conserves angular momentum exactly for spherical symmetrical potentials
- Symplectic, i.e., obeys Liouville's theorem and is area preserving in phase space (this gives them global stability as the energy cannot increase without bound, that would make the area in phase space grow).
- Of better order than euler methods, including Euler-Cromer which has global error of $\mathcal{O}(\Delta t)$ (as it is second-order per timestep).

However, the velocity-verlet is NOT energy conserving (on short time-scales), which the equations of motions should be! Velocity verlet is usually the preferred version of Verlet integration since it has better precision in finite-precision arithmetic

Now, there are methods with smaller errors, such as RK4, but RK4 is NOT symplectic as the determinant of it's jacobian is not 1, therefore RK4 is very risky to use. It is very accurate, so the energy won't grow quickly, but over long time scales, it might grow beyond any bounds and the entire simulation might collapse.

Now, one might question why not turn to a higher-order symplectic integrators? They do exist, for example the Forest-Ruth algorithm which is time-reversible, symplectic and of global order $\mathcal{O}(\Delta t^4)$, the answer is simply that close enough is good enough combined with computational cost. The Forest-Ruth algorithm requires three force-calculations per step, which are very costly. Algorithms such as PEFRL require 4 force calculations per step, a rough estimate then tells us that we can compare PEFRL with a time step four times larger than that of velocity-verlet, PEFRL still comes out much more accurate.

Which method is best depends on use. In some cases, adaptive schemes might be the best.

Cut-off

The most expensive part of MD simulation computationally is the force calculation. The number of pairs in a system grows as $\mathcal{O}(N^2)$, while the cost of the velocity verlet is $\mathcal{O}(N)$ (if we exclude the force calculation). However,

most two-body interactions, such as the LJ potentialare short-ranged, so we can safely ignore many of these force calculations. By looking at the LJ potential, we can see that when r becomes large, $V(r) \approx 0$. We now implement cut-off by defining some distance to big enough, so that we turn $V(r > r_c) = 0$. When we do this, we should shift the entire potential by the constant $V(r_c)$, so that we do not have a discontinious potential.

Now, this means any pair of particles where the distance between them is bigger than r_c can be ignored in the force calculation—the question is just how we do this computationally. If we need to calculate the distance and check, we still have complexity $\mathcal{O}(N^2)$. The answer is neighbor-lists, there are two types: Cell lists, and verlet lists.

In cell lists, we divide the entire system into cubic cells where the width of each box is at least r_c. An atom in a given cell then only needs to consider atoms in their own cells and all the neighboring cells. All other particles are more than one cell away, so there is no force. Figuring out which cell each atom belongs to is obviously $\mathcal{O}(N)$ and must be done every time step. When we know what atoms are in each cell, looping over all neighboring atoms can be done in constant time, assuming the particle density ρ is an intensive quantity, i.e., it does not increase with N—this is usually a very good assumption. Since we can calculate the force on any given atom in constant time, the overall complexity of a force calculation obviously becomes $\mathcal{O}(N)$.

In verlet lists each atom has a list of pointers to their nearest atoms. Any atom closer than $r_c + \epsilon$ is listed. Now, making a verlet list costs a lot, but if ϵ is big enough, we can do several force calculations before reconstructing a verlet list (unlike cell lists). We can combine the two concepts. Using a cell list, we can create verlet lists cheaper.

For L-J, the cutof is usually taken to be $r_{\text{cut}} = 3\sigma$.

Periodic Boundary Conditions

As in any physical problem, we need a set of boundary conditions. There are various posibilities open to us, we could simulate particles in a box (hard walls), in a vacuum (no walls) or periodic boundary conditions (bulk). The particles in a box and vacuum are usually not of that much interest. Due to computational constraints, we cannot really compute for macroscopic quantities of a substance, and so using PBC let us simulate bulk properties of that material without having our results being messed up due to boundary effects.

When we implement periodic boundary conditions, any particle that would leave the box, enters it on the opposite side of the box with its velocity maintained, so that the kinetic energy is maintained. Now, to properly evaluate the potential between pairs of particles, we need to use the minimum image convention. *Images* are the 'copies' of the box we are actually simulating, and so the 'closest' particle might be in the next image, not neccesairly our box.

PBC leads to conservation of linear momentum, but not angular momentum. This is dubbed the 'NVEPG' ensamble or the molecular-dynamics ensamble. It is very close to the micro-cannonical ensamble, but can lead to some small weird artifacts.

Initialization

We also have initial conditions, and we have to be slightly careful with these, however, they are quite forgiving. We usually start out with the atoms a structure grid or lattice. In our project we started of with all the atoms in a face-centered lattice. In addition we give the atoms randomly distributed velocities. What distribution we choose for the velocities will define the total energy of the system. We usually choose a Boltzmann distribution, as this is what we expect the velocities to tend towards anyway. We then eliminate drift by subracting the average velocity from all particles.

When the system is started it *is not* in thermal equilibrium, and there will be drastic changes as the system tends to thermal equilibrium. We call this process equilibriation. If we are going to be looking at a similar system for many different simulation and measurements, we could always store the state of the system after equilibriating it, meaning we can initialize or system very close to a thermal equilibrium in the future.

An important consideration is that the average particle density cannot change within a given simulation (unless of course we change the size of the PBC slowly over time steps), so we should initialize the system with a particle density close to the phase we want to study. If we want to study the gas phase, we shouldn't start the system of as compactly as a solid, as the system has noe room to expand.

Thermostats can be used to bring the system to a target temperature, then turned of.

Efficiency improvements

The cut-off as discussed above is an important algorithmic efficiency improvement. In addition we should rely on Newton's third law, i.e., the fact that $F_{ij} = -F_{ji}$, so that we can get away with only doing each pair of particles once.

Molecular dynamics simulation lend themselves very well to parallelization. Can be done through both OpenMP and MPI.

Other than this there is of course many small C++ improvements to be made, inline functions, using the right data-structures (soa vs aos) and so on. For example, many Armadillo operators are actually quite slow and one should use a quick random number generator.

Saving the thermalized state!

2 Molecular Dynamics in the Microcannonical Ensamble

Discuss initialization and initialization effects. Temperature measurements and fluctuations. Comment on use of thermostats for initialization.

3 Molecular-dynamics in the micro-canonical ensemble

How to measure macroscopic quantities such as temperature and pressure from a molecular-dynamics simulation. What challenges do you expect? What can it be used for?

4 Measuring the diffusion constant in molecular-dynamics simulation

How to measure the diffusion constant in molecular dynamics simulations – limitations and challenges. Compare with methods and results from random walk modeling.

5 Measuring the radial distribution function in molecular dynamics simulations

How can you measure the radial distribution function in molecular dynamics simulations. What does it tell? What challenges will you face? Compare the measurement of the radial distribution function to the measurement of the probability densities for a random walk.

6 Thermostats in molecular-dynamics simulations

Discuss the micro-canonical vs the canonical ensamble in molecular-dynamics simulations: How can we obtains results from a canonical ensemble? Introduce two thermostats, and describe their behavior qualitatively. How can you use such a thermostat for rapid initialization of a micro-canonical simulation?

7 Generating a nanoporous material

- Discuss how we prepare a nanoporous matrix with a given porosity.
- How do we characterize the structure of such a material and the dynamics of a fluid in such a material?

Generation

The main idea of MD simulations on nano-porous materials can be summarized in three steps

1. First we generate a nano-porous matrix (this is the 'solid' material with holes in)
2. Then we fill the holes with a fluid
3. Now we do 'normal MD simulation' on the fluid, focusing on the measurements connected to our interests.

A common example is a matrix Silicon dioxide containing water (H_2O). This method often looks a lot like the one you would follow if doing experimental physics. You prepare a sample in some manner by manipulating temperatures, strains etc. and once we manage to make a material with the right parameters (porosities etc.), we study it's behaviour. For example one could slowly increase the size of the system while simultaneously cool it.

In our simulations, a very important functionality is writing the current 'state' of our entire system to a file. This is important for visualization, but also because we can then continue a simulation or start a brand new one from the saved state. This way we don't have to initialize and thermalize our system for every single run.

The way we generated the system in our project is to first simulate a LJ fluid. After the system has thermalized, we 'freeze' all atoms in given regions, these atoms will no longer be able to move, and are thus part of a 'non-deformable' matrix. The areas not frozen are now the pores in our system, we can remove the atoms in these regions and replace them with an other species.

Characterization and dynamics

The simplest characterisitic is the porosity, ϕ, which is the relative amount of pore space in volume. In my project I simply measured the number of frozen particles and estimated the porosity from the ratio $\phi \approx N_{\text{frozen}}/N_{\text{total}}$.

When filling the pores with a fluid, there are interesting questions that arise—what is the density and pressure of the fluid, do they change with pore size? Can the fluid flow throughout the sample, or are the pores cut off from each other? To characterize the flow, we could for example measure $\langle r^2(t)\rangle$, i.e., the average squared displacement of all fluid atoms in the system as a function of time.

I project 1, we were studying a bulk material, but now we would like to see how pressure is a function of space as well. To do this, we could divide our

system into cells, and calculate the pressure in every cell individually, just like we measured the total pressure in the bulk material earlier. If we use the cell lists, we know that only neighboring cells will affect a given cell by a force, and so this will give the correct pressure.

When I measured the average square displacement, I found that it was a linear function with time, just like normal diffusion outside the nano-porous material. I figure this is because the porosity is high enough, and dependant on how the matrix is made. In a system with many small non-connected pores for example, $\langle r(t)^2 \rangle$ would increase at the start of the simulation but should flatten out as all the fluid atoms in a given pore are restricted to move in a small volume.

Lastly we can measure flow of the fluid through the sample by affect all fluid atoms by an external force—think for example of how gravity causes rain to flow down into sand. We use Darcy's law

$$U = \frac{k}{\mu}(\nabla P - \rho g),$$

where ρ is the density, g is the gravitational acceleration. Since $\rho g = Nmg/V$ we replace ρg with nF where n is the particel density. Darcy law describes the balance between the pressure gradient ΔP and the external force. The constants k and μ are the mediums permeability and the fluids viscosity respectively. And U is the volume-flux (i.e., the volume of the fluid that flows through a given cross-section (for example the boundary) for a given time).

Now, from Darcy's law, we can ignore ∇P in our system, as we don't have hydrostatic conditions, so we have

$$U = -\frac{k}{\mu} n F_x.$$

Now, there are two unknown quantities, so measuring U isn't enough. Now, the permeability k is matrix-dependant and the viscosity μ is fluid-dependant. We can estimate μ for a fluid by measuring the velocity profile of flow through a cylindrical pore, which we expect to have the shape

$$u(r) = \frac{\Delta p}{L} \frac{1}{4\mu}(a^2 - r^2).$$

Where Δp is the pressure difference (at each end of the cylinder) that leads to flow. For us this is $\Delta p = \rho g \Delta h = nFL$.

By simulating the flow through a cylinder, at a stationary state (no acceleration), we can now estimate the fluid viscosity, and we can then estimate the permeability for other systems through Darcy's law by measuring the volume flux. Note that the particle density enters into the viscosity, and so it will be different for a high-density and a low-density fluid of the same species. If we are looking at different nanoporous systems we should fill them with a fluid with the same density if we want to use the same viscosity to estimate the permeability!

8 Diffusion in a nano-porous material

How can you measure the diffusion constant for a low-density fluid in a nanoporous system? Discuss what results you expect. Compare with diffusion in a bulk liquid and in a larger-scale porous medium.

9 Flow in a nano-porous material

Discuss how to induce flow in a nano-porous material. How can you check your model, calculate the fluid viscosity and measure the permeability? What challenges do you expect?

10 Algorithms for percolation systems

- How do we generate a percolation system for simulations?
- How to analyze and visualize the systems?
- How to find spanning clusters and measure the percolation probability?

Generate

We choose some geometry in some dimension, we have usually been studying a $L \times L$ 2D grid. We then let every 'site' in the geometry be either occupied with probability p, or unoccupied with probability $1 - p$. To do this, we draw a random number from $[0, 1)$ for every site (uncorrelated) and check this value against p. Every site with a random number less than p is occupied. This is done very quickly in matlab/python, which creates a random matrix and checks it against some threshold very simple.

Analyze and visualize

We now have our system, which is some matrix of 0's and 1's. We now choose some neighbor-system, we have been working with only closest neighbors (4 neighbors in 2D, not diagonal). Any occpuied site 'in direct contact', meaning you can step from one to the other going only through occupied sites, belong to the same cluster. Our job is thus to go through the matrix, and find all the different clusters. Every cluster is given a unique number label. This can be done quite simply

1. Loop over entire matrix.
2. Skip all unoccupied sites and occpuied sites that have already recieved a label
3. If you hit an occupied site that has no label, label it with current label counter and check for neighbors without labels.
4. If you find a neighbor without label, goto (3). Else: Increase label counter

In practice both matlab and python have good tools built in to do this, `bwlabel` and `measurements.label` respepctively.

Once we have all the labels, we plot the matrix, and give each cluster it's own color. We can either choose color based on cluster-size, or randomize it completely. We should however, *not*, color according to label number. The labeling algorithm cause the label number to increase in a steady fashion, so clusters close together get similar label numbers. In most practical cases, we will get very many clusters, and so the color difference between them will be very small, thus to nearly equal labels will give a color that is almost impossible to tell apart.

We can find the area/mass of any cluster my counting the number of times a given label occurs in the label-matrix.

To find a spanning cluster, we first identify the paired boundaries of our system (often left-right or top-bottom or both). We then find the sets of all labels that occur on either boundary independantly, and intersect the two sets. If a label exist on both boundaries, that cluster is by definition a spanning cluster. For square matrixes, we can also find the minimum bounding box, and checking the width/height against the width/height of the system, matlab and python both have automatic tools for finding bounding boxes of the cluster.

Percolation probability

The percolation probability, $\Pi(p, L)$ gives the probability that a system is percolating.

If we generate a single matrix and check if it is percolating, we of course find that it either is, or it isn't. So if we generate N such systems (same p, same L) we find that n of them are percolating, then the ratio n/N approximates $\Pi(p, L)$. For good statistics, we want N to be as high as possible.

We can of course repeat this for many (p, L) sets to find how the function varies as a function of p and L.

Questions?

If we lock L and vary p, we can use the same underlying matrix and check it versus different p values, we can then find the p' value that makes that exact matrix percolate. Is this a better approach than generating a new random matrix for each p? Why/why not?

11 Percolation on small lattices

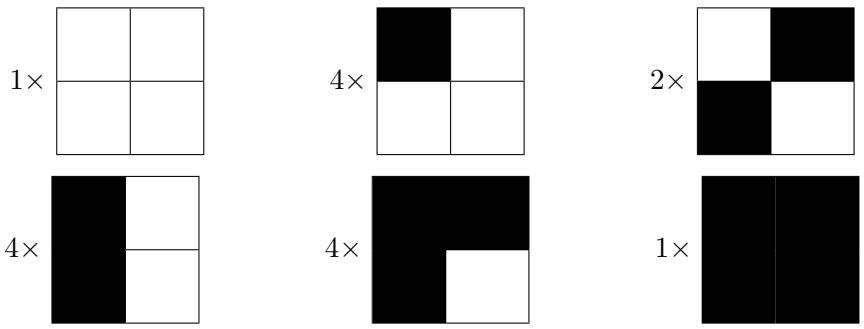
- Discuss the percolation problem on a 2×2 lattice.
- Sketch $P(p, L)$ and $\Pi(p, L)$ for small L.
- Relate to your simulations.
- How do you calculate these quantities and how do you measure them in simulations?

Introduction

We want to find the percolation probability $\Pi(p, L)$ and the spanning cluster density $P(p, L)$ for a $L \times L$ matrix. For small L we can just write out all the possible configurations, write up the probability of them all, and compute the probabilities Π and P on closed form. However, the possible number of configurations grows exponentially with the system size $2^{L \times L}$, however, the insight gained from the small L is very valuable, so let us start by studying a 2×2 and end by relating this to numerical results for higher L.

All configurations

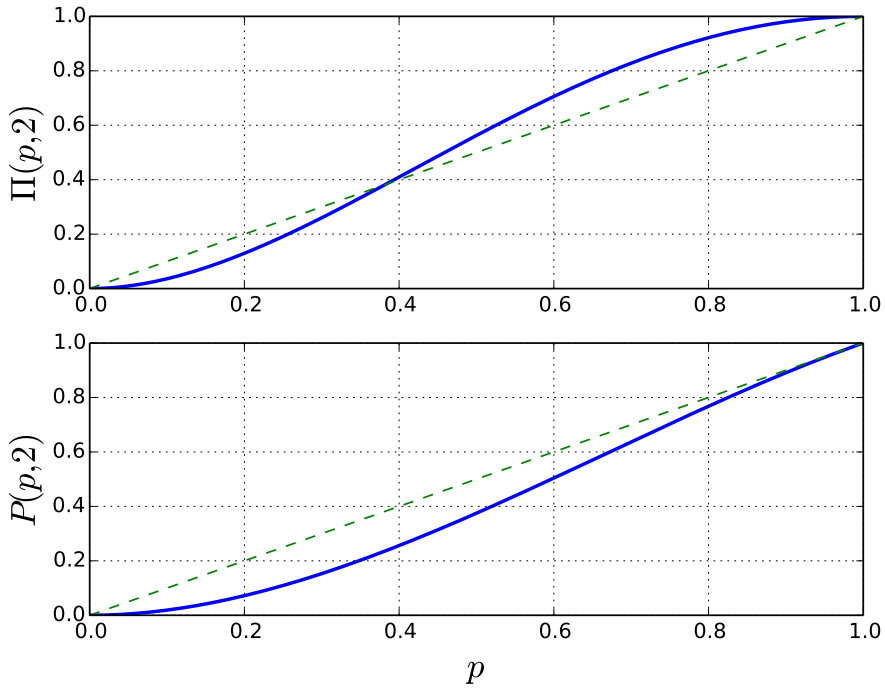
We use nearest neighbor connectivity and consider both left-right and top-bottom to be spanning. There are $2^4 = 16$ possible configurations. We write them all out below:



The probability of any given configuration is given by $p^x(1-p)^{4-x}$ where x is the number of occupied clusters. This gives the expressions

$$\Pi(p, L) = 4p^2(1-p)^2 + 4p^3(1-p) + p^4,$$
$$P(p, L) = 2 \cdot 4p^2(1-p)^2 + 3 \cdot 4p^3(1-p) + 4 \cdot p^4.$$

A plot is shown at the start of the next page



We note some things about the curves: Both curves start at 0 for $p = 0$ and go to 1 as $p \to 1$, this is very intuitive. They are also completely smooth. Note that the spanning cluster density is always below p, this makes perfect sense. The spanning cluster density is the probability of a random site to be a part of the spanning cluster. Now, the random site doesn't *have* to be occupied, so the highest-estimate for P should always be p—and occupied sites don't have to be part of a spanning cluster. In this case, if there *is* a spanning cluster, all occupied sites automatically belong to it, so it is not surprising that the the difference from p dies out as $\Pi \to 1$. For larger clusters however, it is very possible to have both a spanning cluster, and non-spanning cluster in the same experiment.

Measuring $\Pi(p, L)$ and $P(p, L)$ numerically

For larger lattices, listing all the possible configurations becomes impossible due to the sheer number of them. Instead we do monte carlo simulations. By generating N matrices of size L and checking them against the probability p, we can look at the number of them that are percolating, the ratio of percolating systems to the total number of systems simulated gives an estimate for $\Pi(p, L)$ that improves as N improves.

12 Cluster number density in 1-d percolation

- Define the cluster number density for 1-d percolation
- Show how it can be measured.
- Discuss the behavior when $p \to p_c$. How does it relate to your simulations in two-dimensional systems?

Definition of number cluster density

In a random system, there will usually be very many different clusters of varying shapes and sizes. When $p > p_c$ there will usually be one or more spanning clusters, but also many non-spanning clusters. When $p < p_c$ there will usually just be many non-spanning clusters. We might ask the question how the sizes of the clusters are distributed, and how that distribution changes as $p \to p_c$. When $p \to p_c$ the clusters will tend to rapidly grow larger untill they become the system size and the system becomes percolating. The cluster number density $n(s, p)$ is part of this distribution

Definition The cluster number density is the probability for a random site in the matrix to be a specific site in a cluster of size s.

The normalization requirement on the number cluster density is

$$P + \sum_s sn(s, p) = p,$$

In 1D, $p_c = 1$ so $P = 0$ when $p < p_c$ and we get

$$\sum_s sn(s, p) = (1-p)^2 p \sum_s sp^{s-1} = (1-p^2)\frac{\mathrm{d}}{\mathrm{d}p}\sum_s p^s = (1-p^2)p\frac{\mathrm{d}}{\mathrm{d}p}(1-p^2)^{-1} = p.$$

The probability for a random site to be part of a cluster of size s is then $sn(s, p)$, since any cluster of size s has s specific sites. In 1D, for a site to be the left-most site in a cluster of size L is given by the fact that we need p occupied sites and 2 non-occupied sites surrounding those s sites.

$$n(s, p) = p^s(1 - p)^2.$$

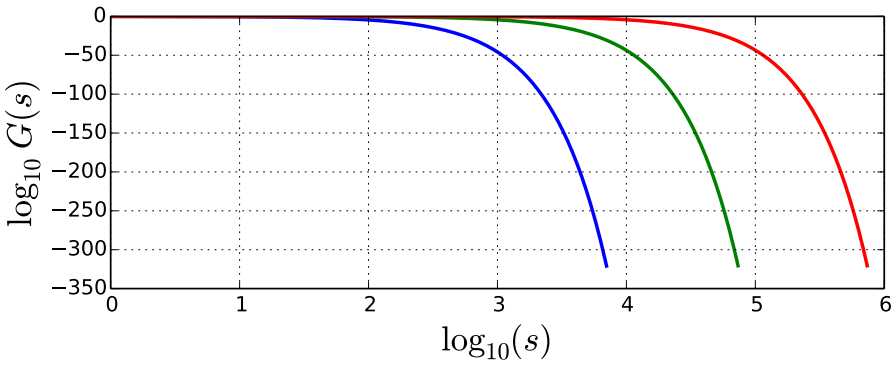
because a random site has a probability p to be occupied, and then it is either part of the spanning cluster, or of a non-spanning cluster of size s.

Characteristic cluster size

To get a better understangin of $n(s, p)$, let us consider p to be a constant. And look at the s-dependance of $n(s, p)$. In this case, $(1 - p)^2$ is basicly just a normalization constant, and of no interest, so we look at only

$$G(s) = n(s, p)(1 - p)^{-2} = p^s.$$

If we plot a log-log plot of this function against s, we get To understand this



function we rewrite G as follows

$$G(s) = p^s = e^{s \ln p} = e^{-s/s_\xi}, \qquad s_\xi \equiv -\frac{1}{\ln p}.$$

Where s_ξ is the cut-off cluster size, also referred to as the characteristic cluster size. It is a cut-off, as

$$G(s) = e^{-s/s_\xi} = \begin{cases} 1 & s \ll s_\xi, \\ 0 & s \gg s_\xi, \end{cases},$$

so when $s \to s_\xi$, G (and thereby $n(s,p)$) starts decay exponentially.

Dependence on p

We see that the p-dependence enters as p affects how large the characteristic cluster size is

$$s_\xi = -\frac{1}{\ln p},$$

from this, we see that s_ξ grows with p (in 1D), which is very intuitive. As $p \to p_c (= 1)$ we can Taylor-expand

$$-\frac{1}{\ln p} = -\frac{1}{\ln(1-(1-p))} \simeq \frac{1}{1-p}.$$

And as $1 = p_c$ we get

$$s_\xi \simeq \frac{1}{p_c - p} = |p - p_c|^{-1/\sigma}.$$

Which is the general form of s_ξ in *any* dimension. In 1D, $\sigma = 1$.

So as $p \to p_c$, s_ξ diverges as a power law.

Data collapse: As the only p-dependence of $n(s,p)$ enters through s_ξ, we expect a data collapse if we instead plot G as a function of s/s_ξ. To include the normalization $(1-p)^2$ we use the Taylor expansion, so that $(1-p)^2 \simeq s_\xi^{-2}$. So that we get

$$n(s,p) = s^{-\tau} F\left(\frac{s}{s_\xi}\right).$$

Numerical simulation

We can generate a system with threshold p, and then count the number of clusters of size s: N_s (the spanning cluster should not be included!). We can then estimate the cluster number density from

$$n(\bar{s},p) = \frac{N_S}{L^d}.$$

A better estimate would of course be given by doing many experiments

$$n(\bar{s},p) = \frac{N_S(M)}{ML^d}.$$

To do the counting $N_S(M)$ in a meaningful manner, we should use logarithmic binning as we have a fat tail.

13 Correlation length in 1-d percolation

- Define the correlation length ξ for 1-d percolation.
- Discuss its behavior when $p \to p_c$.
- How is it related to cluster geometry and your results for twodimensional percolation?

Correlation length ξ

The characteristic cluster size s_ξ says something about the size/mass/area of a cluster. The correlation length however, is used to characterize the extent of a cluster. We first define the correlation function $g(r)$, which is the probability of two occupied sites being a distance r apart of begin connected—the r coordinate is an integer denoting the number sites in between. In 1D, this is quite simple to compute, it is simply

$$g(r) = p^r = e^{r \ln p} = e^{-r/\xi}.$$

This function is quite flat for low r, but as $r \to \xi$ it starts decaying exponentially. As $\partial \to p_c$ the correlation length diverges. We use the fact that $\partial \to 1$ to Taylor expand

$$\ln p \simeq -(1-p)$$

so we have

$$\xi = \xi_0 (p_c - p)^{-\nu}.$$

Where ξ_0 and ν depends on the system, such as dimensionality etc. We note that ξ diverges as a power law as $p \to p_c$.

Cluster geometry

Now, the characteristic length says something about the extent of the cluster, but what about it's geometry? We can define a variance for the geometry of the cluster, such a quantitiy is known as the *radius of gyration*. We define it as

$$R_i^2 = \frac{1}{s_i} \sum_{n=1}^{s_i} (\vec{r}_i - \vec{R}_i)^2.$$

Where R_i is the radius of gyration for a given cluster i, s_i is the cluster size, n is a site label, $\vec{r}$ is the site location and $\vec{R}_i$ is the clusters centre of mass.

2D percolation

For 1D, the extent and mass of a cluster is the same thing (a cluster can't be hollow), so $s_\xi = \xi$, however, in higher dimensions

$$s_\xi \propto \xi^D, \qquad \text{where } D \text{ is a dimensionality constant } D < d.$$

An intersting use for ξ is that it tells us when a finite system size will be a good approximation to an infite lattice. As we have done numerical modeling of a

2D lattice (it is not analytically solvable), we must neccesarily have a finite L, which might affect our results dramatically. However, if $\xi \ll L$, there should be no clusters of extent close to the lattice size, as the clusters of size larger than ξ are exponentially surpressed. However, if $\xi \gg L$ there's a high probability of clusters the size of the system, and so the finite system size will definitely affect the results. As ξ diverges as $p \to p_c$, a finite system size L will never be a truely good approximation to the infite cluster as $p \to p_c$.

We have

$$s_\xi \propto |p - p_c|^{-1/\sigma},$$
$$s_\xi \propto \xi^D = |p - p_c|^{-D\nu}, D\nu\sigma = 1.$$

Finite Size Scaling

14 Cluster size in 1-d percolation

- Introduce the characteristic cluster size for the 1-d percolation problem.
- Discuss their behavior when $p \to p_c$.
- Relate to your simuluations on 2-d percolation.

15 Measurement and behavior of $P(p, L)$ and $\Pi(p, L)$

- Discuss the behavior of $P(p, L)$ and $\Pi(p, L)$ in a system with a finite system size L.
- How do you measure these quantities?

Behaviour

Measurement

16 The cluster number density

Introduce the cluster number density and its applications:

- Definition
- Measurement
- Scaling
- Data-collapse

17 Finite size scaling of $\Pi(p, L)$

Discuss the behavior of $\Pi(p, L)$ in a system with a finite system size L. How can we use this to find the scaling exponent ν, and the percolation threshold, p_c?

18 Subsets of the spanning cluster

Introduce and discuss the scaling of subsets of the spanning cluster. How can we measure the singly-connected bonds, and how does it scale?

19 Random walks / Flow in a disordered system

Either: Discuss the scaling theory for the distance $r^2(t)$ of a random walker dropped at a random occopied site on either (1) the percolation system or (2) the spanning cluster. (You can choose which case to discuss). Relate the results to diffusion in a bulk fluid and in a nanoporous system.

Or: How do you measure the conductivity of the spanning cluster? Discuss the scaling theory for the conductivity $\sigma(p, L)$

Appendix

Box-Muller transform

The Box-Muller transform is a way to convert randomly drawn numbers from $[0, 1)$ to standard normally distributed numbers. There are two versions of the Box-Muller transform, the basic and the polar. Both versions draw to independant uniformely random numbers, u and v from $(0, 1)$ and generate two normal numbers at once using trigonometric functions. The difference is that the polar form implements rejection sampling. If the random numbers u and v are such $u^2 + v^2 \geq 1$, we reject the pair and draw two new ones. Initially this seems inefficient, (we need roughly 1.3 input numbers per output number) but the computation from u and v to normal numbers is computationally cheaper and for most modern computers trigonometric functions are much more expensive than drawing rand numbers.

Questions

Are the signs in Darcy's Law as given in project 2 opposite? Look at wikipedia. Why would the volume flux be *opposite* of the applied force? The law now says that flow goes *against* the pressure gradient (from low pressure to high pressure).