TFY4235/FY8904 : Computational Physics (spring 2019)

Assignment 1: Percolation

Arnau Sala

Department of Physics, Norwegian University of Science and Technology (NTNU), Trondheim

Soft deadline: Feb 14, 2019

Abstract

In this assignment you will study percolation theory: the study of the formation of clusters in a graph. This is the simplest model displaying a phase transition and, due to its simplicity, is one of the best studied problems in statistical physics. Here you will learn how to use random numbers, how to study infinite systems with finite resources (and in a short time!) and much more.

Relevant fields: statistical physics, phase transitions, networks.

Mathematical and numerical methods: finite-size scaling, random numbers, probability density functions, recursive functions, data analysis, root-finding algorithms.

Contact: arnau.sala@ntnu.no; mohammad.alidoust@ntnu.no; ingve.simonsen@ntnu.no

1 Introduction

Imagine you are working in a governmental health care organization and you have to deal with a contagious disease that is spreading along the country. Your role is to decide whether to invest millions in finding a vaccination or wait until the disease dies out, but... will the disease die out? How fast is it spreading? How much time do you have until all the population is infected? Knowing that each person interacts with k other people every day on average and that the probability of infection from an infected individual to a sane one is p, can you tell if the disease will end up infecting every one?

Percolation theory studies this problem and many other. Percolation is the study of the formation of clusters (groups of infected individuals) in networks (groups of people that interact with each other). Another example could be the spread of a fire in a forest, where the probability p will depend on how close the trees are to their neighbors. Percolation also studies the properties of disordered media and complex systems such as the structure of Internet, commerce networks, neural networks (and other biological systems), galactic structures, liquid-gas transition, supercooled liquids, etc.

Before giving more details let me introduce some concepts: a network is a physical system composed of sites (or nodes) and bonds. A square lattice, for example is a network where each site has a neighbor site above, another one below and two more on both sites. The bonds are links that connect different sites. Two sites are neighboring sites if they are connected by a bond. In a square lattice, each site is connected the four closest sites via four bonds.

There are two main types of percolation:

1. **Site percolation:** In a network composed of N sites and M bonds two neighboring sites belong to the same cluster if they are both *activated*. A random node can be activated (or not) with a probability p, the control parameter of the system.
2. **Bond percolation:** In a network composed of N sites and M bonds, two sites belong to the same cluster if there is an activated bond between them or there exist a group of activated bonds that continuously connects them. A random bond is activated with probability p, and this is the control parameter of the system.

Both site and bond percolation are the simplest models displaying a second-order phase transition.

Here we will study the phase transition between the ordered (one big cluster) and the disordered phases (several small, unconnected clusters) in bond percolation. Specifically we want to see how some quantities (P_∞, $\langle s^2 \rangle$ and χ, to be defined later) behave at the phase transition. Around this point these quantities are well characterized by a set of critical exponents (β, γ,...) and our goal will be to find them and also find the point p_c at which the phase transition occurs.

The problem of percolation is a computationally hard problem, but there are some algorithms that are quite efficient. Here we will use an algorithm proposed by Newman and Ziff in 2001 [1]. The details of this algorithm are explained below. If you want to know more about percolation theory take a look at Ref. [2], and if you want to know more about its applications you can read e.g., Ref. [3].

2 The lattice

The phase transitions we want to study are only well defined in infinite-sized systems but since our resources are rather limited we will only study small sized networks. Nevertheless, in order to avoid boundary effects we will impose periodic boundary conditions in our lattices.

The first step of the algorithm consists in generating a 2-dimensional lattice containing N sites. This lattice could be an $N \times N$ matrix with elements

$$a_{i,j} = \begin{cases} 1 & \text{If the site labeled } i \text{ is connected to the site } j \\ 0 & \text{Otherwise} \end{cases}, \tag{2.1}$$

But such a matrix will contain many zeros and will thus not be an efficient implementation. The most efficient implementation for the problem at hand (bond percolation) consists in generating a list of all the bonds of the network. The list should contain pairs of nodes connected by a bond but without repetitions.

Although the lattice should be 2-dimensional, it is more efficient to give one single index to each node: from 1 to N.

Task 2.1: Write a program that, given a number of sites $N = L \times L$ returns a list of pairs of nodes connected by a bond in a square lattice of lateral size L and periodic boundary conditions. Write the list into a file. For any site i the program should return:

$$\begin{array}{ll} i & i+1 \\ i & i+L, \end{array} \tag{2.2}$$

for $i+1$ this should be

$$\begin{matrix} i+1 & i+2 \\ i+1 & i+L+1, \end{matrix} \tag{2.3}$$

and so on. These are the sites at the right and below. By including only two of the four neighbors we will avoid repetition. For the nodes at the rightmost column and at the last row you can make use of the `modulo` function to connect the site i to the site $i-L+1$ if there are no more sites on the right of site i or to the site $i-N+L$ if there are no more sites below.

You can consider elliptical boundary conditions instead, where a site i at the end of the row is connected to the site $i+1$ and to the site $i+L$; and the site N is connected to the site 1 and L.

Task 2.2: How many bonds are contained in a square lattice of size N? And a triangular lattice, where each node is connected to six other nodes? And a honeycomb lattice, where each node is connected to three different nodes?

Task 2.3: Create a program that generates the triangular and honeycomb lattices. Be careful with the periodic boundary conditions on these two lattices. It will also be convenient to write the number of nodes and the number of bonds of each lattice in the same file, in e.g. the first line.

3 Percolating the system

The steps of the program will be the following. In this section we will explain each of these steps in more detail:

1. The program reads the list of bonds from a file and keeps that list in an array.
2. Prepare the system in its initial state: all nodes are clusters that contain only one element.
3. Activate one bond randomly chosen from the list of bonds. If the two sites that are now connected with a bond already belong to the same cluster don't do anything. Otherwise, merge them into the same cluster.
4. Repeat until all the bonds of the list have been activated.
5. Run the program several times to calculate averages. The more times you run the program, the more accurate will be your results.
6. Finally you have to calculate the convolution of your results with a binomial probability distribution function.

During the simulation we have to keep track of some variables such as the size of the largest cluster, the average size of the other clusters and the susceptibility.

3.1 Shuffle the list of bonds

In this section we describe the content of the main program, but each function or routine must be tested separately to make sure that when put together everything will work as expected.

We have already mentioned that we have to activate the bonds into the lattice one by one randomly. One strategy for achieving this goal consists, e.g. in shuffling the list of bonds and then activate them following the new order.

Task 3.1: Create a program that reads the elements in the file where you kept the list of bonds and keeps them in an array of size $(M, 2)$, where M is the number of bonds of the system. It would also be convenient to get from that file the number of sites of the lattice N and the number of bonds M.

In order to shuffle the list we will need to generate random numbers. Many compilers implement their own random number generator, but you can also use a different one. In any case it is very important to initialize the random number generator to a known state: if something doesn't work as expected it will be much easier to find the errors in your code if you can reproduce them again.

Task 3.2: In the same program initialize your random number generator to any *seed* you like and generate 10 or more random numbers. Check that they are indeed different. Now try to run the program again with the same seed. Do you get exactly the same numbers? Do you get different numbers if you use a different seed? If the answer to both questions is yes, then you can proceed to the next task.

Task 3.3: Shuffle the list: take a random number r uniformly distributed between 2 and M and swap the first and the r-th bonds (remember that you have two elements in each row of that list: both should be moved at the same time). Next take another random number r between 3 and M. Now swap the second and the r-th bonds. Repeat it M times.

You can (and must) check that it works for a small list. Make sure that each bond links the same sites as before and that the list is shuffled.

3.2 Prepare the network...

Every time we activate a bond from the list two sites will be connected. We will have to keep track of the state of the network and update this information at each step.

At the beginning all nodes are one-node clusters and when we activate the first bond two clusters merge to form a bigger one. In this new cluster one of the nodes will be the *root* node and the other one will be just a regular node. The information we have to keep is whether any given node is a root node or not, the number of nodes in the cluster and the index (the position, or the name) of the root node: the root node uniquely identifies each cluster.

Task 3.4: Create an array of dimension N (let's call this $sites(i)$). This array will contain the status of each node. If the node i is the root node of a cluster then the i-th element of the list will contain the number of nodes in the cluster. If it is not, then it will contain the index of the root node of the cluster it belongs to.

Since we will keep both indexes and sizes in the same array we can make the sizes negative and indexes positive, so if the i-th element in the list is negative ($sites(i) = -n$) we will immediately know that this is the size of the cluster and that this node is a root node.

At the beginning all sites are root nodes of one-node clusters, so all the elements of this array should be initialized to -1.

3.3 ...and start activating bonds

Take elements of your shuffled list of bonds (take the indexes of the two sites) and activate them one by one. If the two nodes already belong to the same cluster don't do anything. If they belong to different clusters then update the information of the network: the root node of the smaller cluster now points to the root node of the largest cluster and the cluster size is also increased.

In order to do this efficiently we have to create a function that given the index of a site j returns the position of the root node of the cluster this site belongs to. For this you can use a recursive function.

Task 3.5: Write a function that, given a node j, if j is a root node ($sites(j) < 0$) then it returns j, and if it is not then the function calls itself again with the argument $sites(j)$ while, at the same time updates the information of $sites(j)$, i.e., now $sites(j)$ will point directly to the root node.

If this is the first time you use a recursive function you may want to play with it, but be careful: when using recursive functions you can easily run into a stack overflow (in UNIX systems the size of the stack is usually limited to $\sim$8 MB, and with Python this is further limited to $\sim$1000 recursion steps, but of course you can always increase it). For our program this is not a problem because if we update the information in $sites(i)$ every time we look for a root node, we will need only three calls of that function on average, even if your system consists of more than 10^6 sites.

Task 3.6: Let us go back to the list of bonds: take one bond (two sites i_1 and i_2). Call the function to find their corresponding root nodes r_1 and r_2. If $r_1 = r_2$ the two nodes already belong to the same cluster and no further actions are required. If they are not the same and $sites(r_1) < sites(r_2)$ (cluster 1 bigger than cluster 2, remember the size is negative) then link r_2 to r_1 and update the cluster size in r_1.

Task 3.7: Keep track of the largest cluster. Each time you merge two clusters or just add a new site compare the size of the new cluster with the largest one in the previous iteration. Keep the index to the root node of the largest cluster.

Now Take a "picture" of the system every few steps. Each time you activate a random bond to the network you are creating a network where the probability of activating a node is given by

$$p = \frac{\text{Number of activated bonds}}{\text{Total number of bonds}}. \tag{3.1}$$

Our goal is to find the probability p_c at which the system percolates, i.e. a giant cluster spans across the whole network. Let us try to find this point "graphically" first and then we will do a more accurate search.

Task 3.8: For four or five different values of the probability p in the range (0,1) make a list containing only the nodes that belong to the main cluster. Note that when you merged two big clusters the nodes of the smaller cluster pointed to the root of the smaller cluster, so looking at $sites$ to see which nodes belong to the largest cluster is not enough. You have to use the function that finds the the root nodes on each site and add to the list the nodes that have the same root.

Task 3.9: Now take that list and transform it into a physical lattice: plot the position of each site in a 2-dimensional space. Assign to each node a coordinate x and y and plot the results

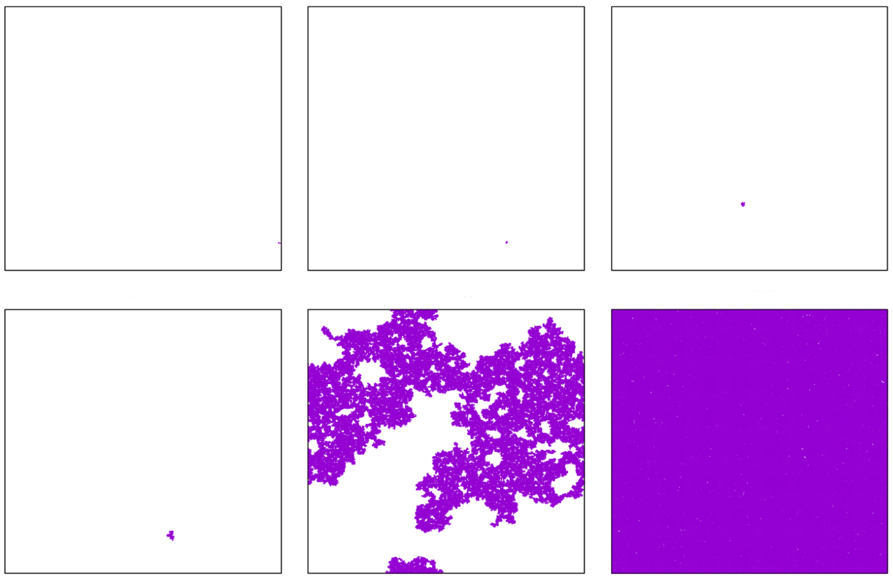

Figure 1: Different "pictures" of the network taken at different p. These pictures show a clear phase transition between the unpercolated phase and the percolated one, where a giant cluster spans across the network.

for the different p. Can you see the phase transition? Give an estimate of the probability p_c (doesn't have to be very accurate, you will get accurate results later on). You should get something similar to Fig. 1.

3.4 Measurements

These pictures give you a graphical description of what is happening but they are not part of the algorithm. We have to find more accurate methods to calculate the probability p_c. In order to study the phase transition we will keep track of three variables:

1. P_∞, also called "giant component". This is the relative size of the largest cluster. It is defined as the number of sites in the largest cluster divided by the total number of sites. Each time you activate a bond in the lattice you have to update this quantity.

2. $\langle s \rangle$, the average cluster size. Actually we are interested in the weighted average cluster size [2]: $\sum_i s_i^2/(N - s_\infty)$, where the sum runs over all the clusters except the largest one, s_i are the sizes of the different clusters and in the denominator s_∞ is the size (absolute) of the largest cluster.

3. χ, the susceptibility, defined as $\chi = N\sqrt{\langle P_\infty^2 \rangle - \langle P_\infty \rangle^2}$.

Keep all this data in three different arrays (or one matrix) with M components. At the end of this calculation we will need the value of these variables as a function of the probability p, which will have M different values.

Task 3.10: Complete your code by adding some lines to keep track of these quantities.

Measuring P_∞ and P_∞^2 is easy: if you kept track of the root node r_1 of the largest network you just have to measure $|sites(r_1)|/N$ and $(sites(r_1)/N)^2$, respectively. Keep in mind that you may have to run this part of the code few times to make averages over P_∞, $\langle s \rangle$ and χ, so program consequently (define and initialize variables outside the loop and add them up inside). For the average cluster size this is more tricky. What I believe is the most efficient way consists in the following:

1. Create a variable to store the sum of the size squared of all the clusters (e.g. *average_s*). At the beginning, before activating any bond, all N sites are clusters of size 1, so $average\_s = N$.

2. Now you can start iterating over the list of bonds. Every time you activate a bond you have to update *average_s*. Once you activate a bond this will link two sites s_1 and s_2. If these two sites already belong to the same cluster (they are connected indirectly by other bonds) then the distribution of clusters remains unchanged and no actions are required. But if s_1 and s_2 belong to different clusters you have to (i) subtract from *average_s* the square of the size of the two clusters you are going to merge—that is $sites(r_1)^2$ and $sites(r_2)^2$, with r_1 and r_2 the root nodes of the cluster that contain s_1 and s_2, respectively—, (ii) add the square of the size of the new merged cluster to *average_s*.

3. Then you can calculate $\langle s \rangle$ for each probability p as

$$\langle s \rangle = \frac{average\_s - (NP_\infty)^2}{N(1 - P_\infty)}. \tag{3.2}$$

   With this you are calculating the average squared cluster size of only the clusters that do not belong to the giant component.

Be careful with infinities! At some point you will have $P_\infty = 1$ and this may make $\langle s \rangle$ diverge, but it should be exactly 0.

Task 3.11: For now one single realization is enough: let's finish the program and test it, but if later on you see that the results are not satisfactory enough you may want to run this part of the code a few times to get averages of P_∞, $\langle s \rangle$ and χ. So this task consists in adding a loop to get these averages.

At each iteration you have to initialize $sites(i)$ and reshuffle the list of bonds (but do not reinitialize the random number generator!). If you want you can make just one iteration and increase the number of iterations only when you are sure that the program works.

4 Convolution

The main advantage of using this algorithm is that you only have to go through all the M bonds of the network once, instead of having to go through the M bonds M times, as it is required by the "classical" algorithms. This means that the time it takes to run this program scales as $\mathcal{O}(M)$ instead of $\mathcal{O}(M^2)$, but to make our results valid we have to embed the results into the right statistical ensemble. In this case this is done by a convolution of our data with a binomial probability distribution function, since there are $\binom{M}{n}$ different ways to activate n bonds on a lattice with M bonds. This step might take some time for large lattices, but in any case, you can get results in few hours $N = 10^6$.

Given a set of measurements $\{Q_n\}$ for $n \in \{0, \dots, M\}$—such as $\{P_\infty(0), P_\infty(1), \dots, P_\infty(M)\}$, the giant component after activating each of the M bonds—, their corresponding values as a function of a probability q are given by

$$Q(q) = \sum_{n=0}^{M} B(M, n, q) Q_n = \sum_{n=0}^{M} \binom{M}{n} q^n (1 - q)^{M-n} Q_n. \tag{4.1}$$

The following tasks consist on calculating this quantity for the different data you have been collecting in your program.

Task 4.1: How can you calculate the coefficients of the binomial? Remember that M is a very large number. Can you calculate $M!$? Try it with $\log(M!) = \sum_i \log i$. Or you can simply use an iterative algorithm to find the coefficients of the binomial (Hint: $\binom{M}{n} = \frac{M-n+1}{n}\binom{M}{n-1}$). If the numbers are still too big you can calculate the logarithm of each factor in Eq. (4.1) and then take the exponential of the whole expression.

Task 4.2: You will use these coefficients many times inside two loops (one for n and one for q), and calculating them take time... Why don't you calculate them before starting the loop and keep them in an array? This will make your program much more efficient.

Task 4.3: Calculate the convolution to obtain $P_\infty(q)$, $P_\infty^2(q)$ and $\langle s\rangle(q)$ and save the data in a file. We will study the behavior of these quantities during a phase transition and, as you will see in the next section, $\langle s\rangle$ and χ diverge at $q = p_c$, so make sure you have enough data points to study its behavior with enough resolution. I recommend you to use at least 10^4 steps in q between 0 and 1, but if your program takes too long to go through this task you can use 1000 steps[1].

Making the system percolate takes a long time, and calculating the convolution too. Be prudent and save all your data in a file once you arrive at this point.

5 Analysis of the results

The first part of this assignment consisted on generating some data, but without a proper analysis this data is meaningless. Here you will extract useful information from the data files that you generated.

Task 5.1: But first you need to collect more data. Run the program different times with lattices of different sizes (N =1000, 40000, 90000, 250000,...). Plot the giant component, the average cluster size and the susceptibility as a function of q for two lattices of different size. Do you see any difference? Why?

Task 5.2: These plots show a phase transition. The two phases are (i) a phase where the system is composed of many very small clusters or just unconnected sites and (ii) a phase where one of these clusters become of the same size as the lattice and spans across the whole space the lattice is defined on. Can you guess what would be the probability $q = p_c$ at which an infinite sized system percolates?

Percolation, as any statistical physics or thermodynamics problem is a problem involving an infinite number of particles. You may remember from other courses that phase transitions are only well defined in infinite systems, thus to obtain the necessary data to study these phase transitions with enough accuracy (and translate the results to other fields where $\mathcal{O}(N_A)$ particles are involved[2]) we need to study infinite-sized systems. So the next task consists on creating a lattice with $N = \infty$ and repeat the calculations. Can you do that in a finite time? Don't worry, there is trick to study infinite-sized systems: Finite-size scaling.

[1]I did this with python with 1000 steps and $M = 2 \times 10^6$ and it took me 5 hours, but the program could be optimized a lot

[2]A liquid-gas phase transition or percolation through a porous media involve $\sim 10^{24}$ particles or nodes.

Finite-size scaling (FSS) is a theory that makes it possible to analyze the behavior of a system near a phase transition by running simulations on finite systems only. I am not going to enter into details. If you want to know more about FSS in Ref. [2] you will find many very useful information. The basic things you should know are:

1. Near the phase transition some variables, among which P_∞, $\langle s \rangle$, χ and the correlation length ξ (for this assignment it is enough to know that the correlation length is proportional to a typical cluster diameter [2]), scale as a power law of $q - p_c$:
$$P_\infty \propto (q - p_c)^\beta, \tag{5.1}$$
$$\langle s \rangle \propto |q - p_c|^{-\gamma}, \tag{5.2}$$
$$\xi \propto |q - p_c|^{-\nu}. \tag{5.3}$$

2. From these equations you can already see that for $q \sim p_c$, the average cluster size and the correlation length will diverge, but this will be only true for infinite systems. What happens then for finite sized systems? In this case ξ will be as large as the spanning cluster. That is $\xi \sim \sqrt{N}$ (this applies only to two-dimensional systems, for random networks the relation you have to use is $\xi \sim N$, but we will get to this later).

3. Now you can rewrite Eqns. (5.1) and (5.2) as a function of the size of the system. Your equations are now valid for finite-sized systems and you can obtain the critical exponents β, γ and ν, as well as p_c.

Task 5.3: Using Eqns. (5.1) and (5.3) you can get the expression $P_\infty \propto \xi^{-\beta/\nu}$. This expression relates the size of the largest cluster P_∞ to the size of the system $\xi \sim \sqrt{N}$. Since this expression is only valid at the phase transition you can find p_c as the probability that makes P_∞ a power law of the size of the system. You have to plot P_∞ against ξ for each different q in a log-log plot. Fit a straight line to your data and find the probability q that makes the coefficient of correlation R^2 larger. That will be the probability p_c and the slope of the curve will be $-\beta/\nu$.

Task 5.4: $\langle s \rangle$ diverges at the critical point, but this point is different for lattices of different sizes. Plot $max(\langle s \rangle)$ against ξ (again, using $\xi \sim \sqrt{N}$) on a log-log scale to find γ/ν. Finally, the exponent ν can be determined from Eq. (5.3): find $q_{\max}$, the probability at which $\langle s \rangle$ takes its maximum value, and plot $|q_{\max} - p_c|$ against ξ again in log-log scale and fit a straight line to the data points. This will give you $-1/\nu$. Give the three critical exponents β, γ and ν. There are other exponents that characterize a phase transition (take a look e.g., at Ref. [4]). Can you find them all?

5.1 Other 2D lattices

Not all the systems can be modeled as a two-dimensional square lattice, but if you have done it right your program should be able to take any network and and give you the probability p_c and the critical exponents. Let's try it with two simple two-dimensional lattices.

Task 5.5: Generate other lattices (honeycomb, triangular, etc.) and repeat. Do you get the same probability p_c? And the same critical exponents? Why?

Task 5.6: If you have done it right you may have seen that there is a special relation between the critical point p_c for the triangular and for honeycomb lattices. This arises because the triangular lattice is the dual graph of the honeycomb lattice. So studying only one of these two lattices would be enough. Prove that the triangular lattice is the dual graph of the honeycomb lattice. Find also the dual graph of the square lattice. What does this tell you about p_c for a square lattice?

6 Random networks (optional)

Not all the physical systems can be modeled as a periodic lattice. Some systems such as the Internet, the human brain, world trade networks, etc. are complex networks with nodes and bonds distributed randomly. To study the spread of a disease or the distribution of prime numbers over the integers we also need complex networks [3, 5]. So, for those of you interested in complex systems, neurology, biophysics, epidemiology, arithmetics... actually for those of you interested in science, learning about complex networks may result very useful.

In the following tasks you will learn how to generate a random network, that is, a network whose degree distribution (the number of bonds connecting to each site) follows some probability density function (PDF). Generating these networks is not easy: if you just assign bonds randomly to a list of sites you may end up with two or more unconnected graphs. Nevertheless you can still get some results if the largest connected graph is big enough.

Here you will learn how to generate a random network with a given PDF and use your algorithm to study percolation in those networks.

6.1 Random numbers

First we have to create a function that returns random numbers with some PDF. The sites of the network will have a degree distribution (average number of bonds per site) given by some PDF $P(k)$, where k is the degree of the sites.

Task 6.1: Create a function that generate random numbers distributed with (i) an exponential PDF and (ii) a power law PDF. These are given by[3]:

$$P_{\text{Exp}}(k) = p(1-p)^k \tag{6.1}$$

$$P_{\text{P-L}}(k) = \frac{\gamma - 1}{k_0}\left(\frac{k}{k_0}\right)^{-\gamma}. \tag{6.2}$$

To generate exponentially distributed random numbers do the following:

1. Choose an average degree $\langle k \rangle$. Generate also a random number r uniformly distributed between 0 and 1.

2. Calculate $p = 1/(\langle k \rangle + 1)$.

3. Your uniformly distributed random number r is transformed into an exponentially distributed random number r_e as

$$r_e = \frac{\log(1-r)}{\log(1-p)} - 1. \tag{6.3}$$

You can check this using $\sum_{i=0}^{r_e} p(1-p)^i = 1 - (1-p)^{r_e+1}$. Similarly, for a power law PDF do the following:

1. Choose an average degree $\langle k \rangle$. Generate also a random number r uniformly distributed between 0 and 1.

2. Power laws PDF are characterized by an exponent γ (this should not be confused with the critical exponent γ). Set $\gamma = 3.5$ (later on you will see why).

[3]Note that $P_{\text{Exp}}(k)$ is a discrete PDF, while $P_{\text{P-L}}(k)$ is continuous. In the first one k goes from 0 to ∞ taking only integer values and in the second k goes from some cutoff k_0 to ∞ along the reals.

3. Your uniformly distributed random number r is transformed into an power-law distributed random number r_p as

$$r_p = \langle k\rangle \frac{\gamma-2}{\gamma-1}(1-r)^{1/(1-\gamma)}. \tag{6.4}$$

You can check this using

$$\int_{k_0}^{r_p} P_{\text{P-L}}(k)dk = \frac{1}{k_0}\left(k_0 - r_p\left(\frac{r_p}{k_0}\right)^{-\gamma}\right) \tag{6.5}$$

$$\int_{k_0}^{\infty} kP_{\text{P-L}}(k)dk = k_0\frac{\gamma-1}{\gamma-2}. \tag{6.6}$$

You will call these functions with a uniformly distributed random number r and an average degree $\langle k\rangle$ as arguments and these should return a transformed random number.

Task 6.2: Prove that with these two functions you generate random numbers distributed with the PDF in Eqns. (6.1) and (6.2). That is, plot a histogram with the frequency of each degree together with Eq. (6.1) and (6.2).

6.2 Generate the network

Now we can generate networks with N sites and an average degree given by the aforementioned PDFs.

Task 6.3: Create an N-dimensional array $degree(i)$. Here you will write the degree of each site. Set the average degree you would like. Try e.g., $\langle k\rangle = 5$. Using the functions you created in the previous tasks assign a degree to each site (for the exponential network use $degree(i) = \lceil r_e \rceil$, where $\lceil\cdot\rceil$ is the ceiling function, to avoid unconnected sites).

Now we have to link the nodes randomly but respecting their degree. Ideally we would end up with a network containing N sites and $M = \langle k\rangle N/2$ bonds, but we will probably have a few more or less bonds.

Task 6.4: Let's start joining the sites. You have to take every possible pair of sites (let's call them s_i and s_j) and join them with a probability p given by

$$p = \frac{1}{1 + \frac{2M}{degree(s_i)\cdot degree(s_j)}}. \tag{6.7}$$

This can be done with two nested loops:

1. Create a loop that runs from $s_i = 1$ to $N-1$.

2. Inside this loop create another loop that runs from $s_j = s_i + 1$ to N.

3. Generate a random number r uniformly distributed between 0 and 1 and join the two sites if $r < p$ (write them into an array in the same way as you did with the square lattice).

4. Count the total number of bonds: set up a counter before starting the loops and every time you join two sites add 1 to this counter.

At the end you will have a list of pairs of nodes joined by a bond but... where did you keep that? In an array? Did you know the dimension of that array before starting?

Task 6.5: Write the list of bonds in a file. Remember to write the number of sites and the number of bonds that you have measured in the first line of that file. Is this number of bonds close to M?

Task 6.6: Generate random networks with different sizes but always with the same $\langle k \rangle$.

Task 6.7: Run your algorithm to percolate the system on both the exponential and the power law random networks. Find $P_\infty(q)$, $\langle s \rangle(q)$ and measure p_c and the critical exponents. Use the same programs as before, but now use $\xi \sim N$: In a random network the dimension of the system is not well defined, so it is not possible to find the critical exponent ν. Instead we will calculate $\bar{\nu} = d \cdot \nu$. To do so just set $\xi \sim N$ and proceed as in section 5.

Task 6.8: For random networks the probability p_c can be found at

$$p_c = \frac{\langle k \rangle}{\langle k^2 \rangle - \langle k \rangle}. \tag{6.8}$$

Check that your results are consistent with this. For the critical exponents don't worry if they don't agree with what you can find in the literature: In order to get good results in this section you should generate many different networks with the same average degree and the same size and make averages over them. Also you would have to use only networks that are fully connected graphs. But you are not required to do any of that.

Other quantities can also be measured analytically for random networks. Let us take a look at the evolution of the giant component $P_\infty(q)$. This can be obtained with:

$$y(q) = 1 - \sum_{k=0}^{\infty} \frac{kP(k)}{\langle k \rangle} (1 - py(q))^{k-1} \tag{6.9}$$

$$P_\infty(q) = 1 - \sum_{k=0}^{\infty} P(k)(1 - py(q))^k. \tag{6.10}$$

By solving the first equation with some PDF $P(k)$ and placing it into the second you can obtain $P_\infty(q)$ (Note: The sum goes from 0 to ∞, but you can cut it off safely at $k \sim 1000$).

Task 6.9: Solve these equations with a root-finding algorithm and plot the $P_\infty(q)$ you obtained numerically with the one you obtained from these expressions. Do the results agree?

(Note: If you set $\gamma < 3$ in Eq. (6.2) then the second moment $\langle k^2 \rangle$ will diverge and, according to Eq. (6.8), the probability p_c will be 0. This means that the system is always percolated!)

References

[1] M. E. J. Newman and R. M. Ziff, Phys. Rev. Lett. **85**, 4104 (2000), URL `http://link.aps.org/doi/10.1103/PhysRevLett.85.4104`.

[2] D. Stauffer and A. Aharony, *Introduction to percolation theory* (Taylor & Francis, 1992), ISBN 9780748400270.

[3] M. Sahini and M. Sahimi, *Applications Of Percolation Theory* (Taylor & Francis, 2003), ISBN 9780203221532.

[4] *Percolation critical exponents: Scaling relations*, accessed: 28-11-2016, URL https://en.wikipedia.org/wiki/Percolation_critical_exponents#Scaling_relations.

[5] S. M. Dammer and H. Hinrichsen, Phys. Rev. E **68**, 016114 (2003), URL http://link.aps.org/doi/10.1103/PhysRevE.68.016114.