Project 3

FYS4460 - Disordered systems and percolation

Anders Johansson
3rd June 2018

All files for this project are available at `https://github.com/anjohan/fys4460-3`.

a)

As my Easter procrastination was reading a Fortran book[1], I decided to implement this project in Fortran. Since there is no `scipy.ndimage.measurements.label` in Fortran, the most important step was to implement an algorithm for labelling clusters in a binary matrix. Many high-performance algorithms exist for this task, and implementations of several of these in Fortran can be found on the internet. I, however, came up with a low-performance algorithm myself and chose to implement it for fun.

My algorithm goes through the entire grid, and for each site checks if it is occupied and unlabelled. If both these conditions are met, a recursive algorithm grows the entire cluster connected to this site.

The main algorithm is the `label` subroutine,

```
subroutine label_recursively(matrix, labelled_matrix, num_labels)
    logical, dimension(:,:), allocatable, intent(in) :: matrix
    integer, dimension(:,:), allocatable, intent(inout) :: labelled_matrix
    integer, intent(inout) :: num_labels
    integer :: L, i, j

    L = size(matrix,1)
    num_labels = 0

    if(.not. allocated(labelled_matrix)) then
        allocate(labelled_matrix(L,L))
    end if

    labelled_matrix = 0

    do j=1,L
        do i=1,L
            if(matrix(i,j) .and. labelled_matrix(i,j) == 0) then
                num_labels = num_labels + 1
                call growcluster(matrix,labelled_matrix,i,j,num_labels)
            end if
        end do
    end do
end subroutine
```

where the recursive `growcluster` subroutine simply checks if each neighbouring site is occupied and unlabelled,

```
recursive subroutine growcluster(matrix, labelled_matrix, i, j, label)
    logical, dimension(:,:), allocatable, intent(in) :: matrix
    integer, dimension(:,:), allocatable, intent(inout) :: labelled_matrix
    integer, intent(in) :: i, j, label
    integer :: L

    L = size(matrix,1)
```

```
        labelled_matrix(i,j) = label

        if(i<L) then
            if(matrix(i+1,j) .and. labelled_matrix(i+1,j)==0) then
                call growcluster(matrix, labelled_matrix, i+1, j, label)
            end if
        end if
        if(j<L) then
            if(matrix(i,j+1) .and. labelled_matrix(i,j+1)==0) then
                call growcluster(matrix, labelled_matrix, i, j+1, label)
            end if
        end if
        if(i>1) then
            if(matrix(i-1,j) .and. labelled_matrix(i-1,j)==0) then
                call growcluster(matrix, labelled_matrix, i-1, j, label)
            end if
        end if
        if(j>1) then
            if(matrix(i,j-1) .and. labelled_matrix(i,j-1)==0) then
                call growcluster(matrix, labelled_matrix, i, j-1, label)
            end if
        end if
    end subroutine
```

This algorithm is easily extensible to other geometries and connectivities, as the basic idea "check all sites, check all neighbours" is generally valid. Higher dimensions should also be unproblematic, although a generalisation of checking all neighbours would be useful to avoid tedious coding.

There is, however, one pressing issue with this algorithm: It is recursive (and not tail-recursive). As a result, it will give a stack overflow for sufficiently large simulations. For this reason, I reluctantly had to abandon my algorithm and instead write a small Fortran interface for a C implementation of the Hoshen-Kopelman algorithm found on the internet[3]. Unfortunately, this implementation uses a lot of global variables and is not thread safe, so I now have my own Fortran implementation.

Spanning cluster detection is done by checking if either the top and bottom rows or the left and right columns contain common elements. This is implemented by first going through one of the arrays and registering which labels are present, and then going through the other array and checking if any of the labels in this array were also present in the first,

```
        label_found(0:num_labels) = .false.

        do i=1,L
            label_found(array1(i)) = .true.
        end do

        do i=1,L
            if(array2(i) /= 0 .and. label_found(array2(i))) then
                intersect_label = array2(i)
                return
            end if
        end do
```

The algorithm assumes that there is only one spanning cluster. This should be unproblematic, as it is

highly unlikely that two clusters stretch from one side of the grid to the other without touching each other and thus being the same cluster. When the label of the spanning cluster has been found, the area is straightforwardly calculated by a command like `count(labelled_matrix == spanning_label)`. A sample run gives

```
Binary matrix:
   F   F   T   F   F   T
   F   F   T   T   F   T
   T   F   T   F   T   T
   T   T   T   T   F   T
   F   T   T   T   T   T
   F   T   F   F   F   T
Labelled matrix:
   0   0   1   0   0   1
   0   0   1   1   0   1
   1   0   1   0   1   1
   1   1   1   1   0   1
   0   1   1   1   1   1
   0   1   0   0   0   1
Spanning cluster: 1
Spanning cluster area: 21
```

The functions $\Pi(p, L)$ and $P(p, L)$ are estimated by repeating a simulation multiple times. $\Pi(p, L)$ is the number of repetitions which lead to a spanning cluster divided by the total number of repetitions, while $P(p, L)$ is estimated by the average size of the spanning cluster (or zero if there isn't one), divided by the total size of the grid, L^2. Averaging over many cycles is trivially parallelisable with OpenMP.

b)

The probability of having a spanning cluster, $\Pi(p, L)$, and the density of the spanning cluster, $P(p, L)$, were estimated with the Fortran machinery described in the previous exercise. $\Pi(p, L)$ is calculated by creating many random, binary matrices, counting the number of matrices which have a spanning cluster and dividing by the total number of matrices created.

If there is no spanning cluster, $P(p, L) = 0$, otherwise it is equal to the number of sites belonging to the spanning cluster, divided by the total number of sites. Hence it is equal to the probability of one randomly selected site being on the spanning cluster. An estimate is found by creating and labelling many binary matrices and averaging the number of sites on the spanning cluster (or zero), and dividing by the total number of sites on the lattice.

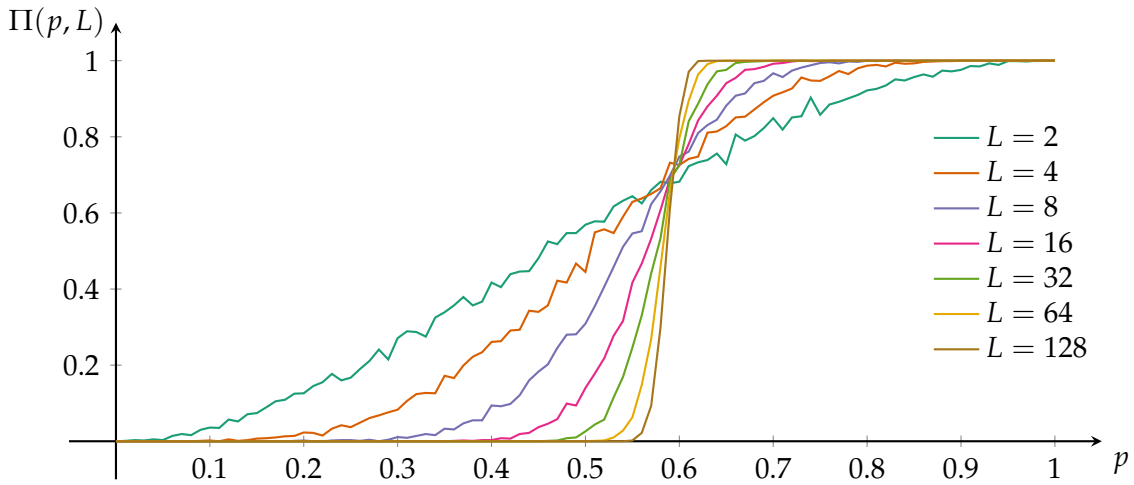
Mathematically,

$$\Pi(p, L) = \frac{\sum_{\text{cycles}} \begin{cases} 1, & \text{spanning cluster exists} \\ 0, & \text{no spanning cluster exists} \end{cases}}{\text{number of cycles}} \tag{1}$$

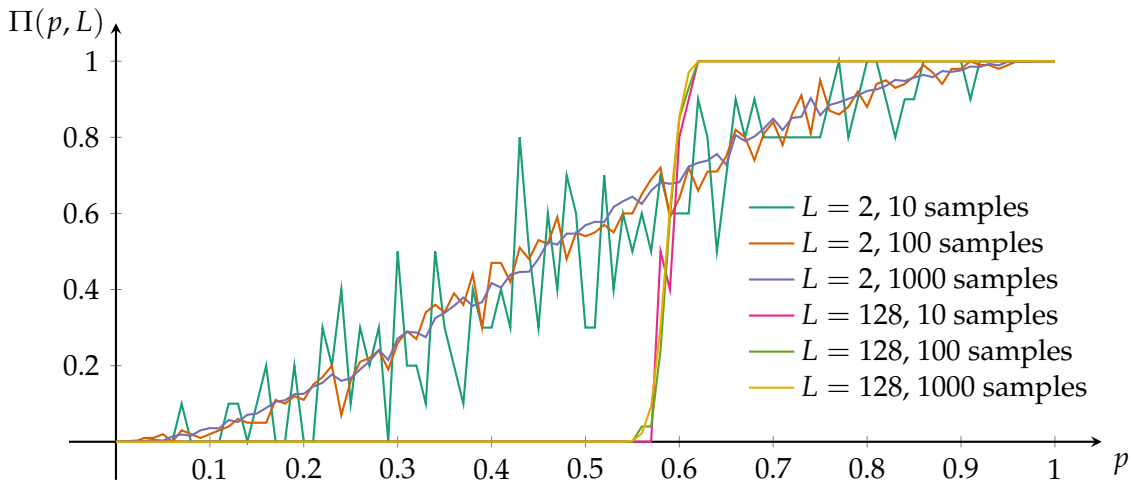
and

$$P(p,L) = \frac{\sum_{\text{cycles}} \begin{cases} 0, & \text{no spanning cluster exists} \\ & \text{mass of spanning cluster} \end{cases}}{L^2 \cdot \text{number of cycles}}. \tag{2}$$

A variety of system sizes L and numbers of samples were used for probabilities in the range from zero to one. See figure 1 and figure 2 on the next page.



(a) Different system sizes. As L increases, the function shape approaches a step function, with a clearly defined critical probability p_c. Averaged over 1000 Monte Carlo samples.



(b) Different numbers of Monte Carlo cycles for the largest and smallest system sizes simulated.

Figure 1: Probability of having a spanning cluster, compared both for different system sizes and for different numbers of Monte Carlo cycles. The graphs show that large systems give good results with very few samples, while small systems require a large number of samples to give good results with little noise.

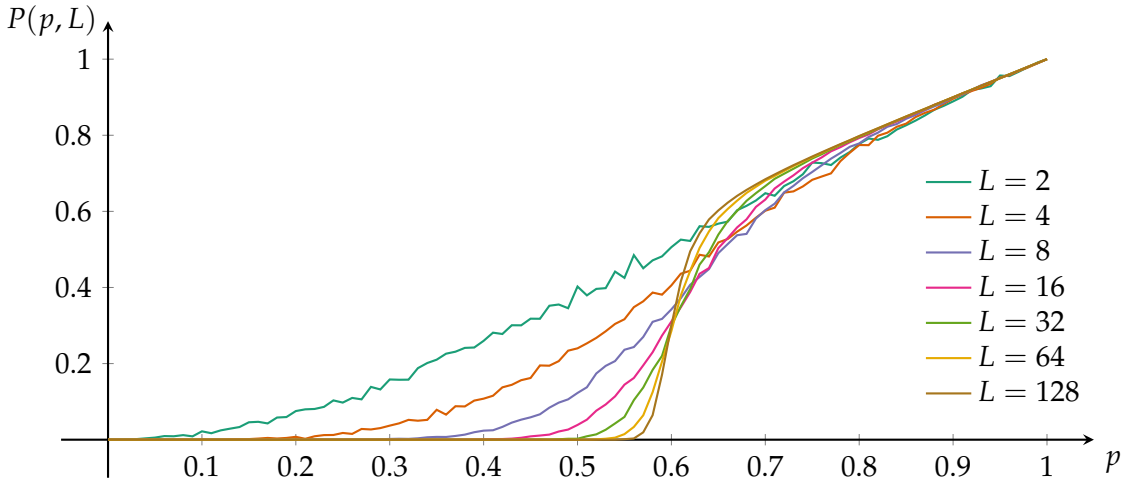


Figure 2: Spanning cluster density as a function of the probability p for a site to permit transport, for several system sizes, with 1000 samples. For large systems, the density has a clear jump when it becomes probable to have a spanning cluster. Beyond this value, the density increases linearly as the size of the spanning cluster increases.

c)

The same functions as in the previous exercise were used to calculate $P(p, L)$ for probabilities slightly above the critical probability. A lower bound of exactly p_c makes the computer complain when attempting to calculate $\ln(p - p_c)$.

According to the exercise description, the spanning cluster density $P(p, L)$ should be proportional to $(p - p_c)^\beta$ for some exponent β for $p > p_c$. The exponent is found by doing linear regression on a log-log scale, as

$$P(p, L) = a(p - p_c)^\beta \implies \log_{10}(P(p, L)) = \log_{10}(a) + \beta \log_{10}(p - p_c). \tag{3}$$

The exponent is therefore the slope of $\log_{10}(P(p, L))$ as a function of $\log_{10}(p - p_c)$. Figure 3 on the following page shows the simulation results together with the theoretical model with $\beta = 0.195$ found from linear regression.

Fortran does, unfortunately, not have an `np.polyfit` function, so the linear fit was done manually by calling a least squares subroutine from LAPACK. Figure 3 on the next page shows the result.

A fairly poor fit is found for probabilities which are either close to or far away from the critical probability. The error for probabilities close to the critical probability is due to finite size effects; the theoretical model is exactly zero, while the finite size of the lattice causes the spanning cluster density to become non-zero even when p is slightly smaller than p_c. On the opposite side of the spectrum, the deviation is also large when $p \gg p_c$ due to the limited range of validity for the theoretical model.

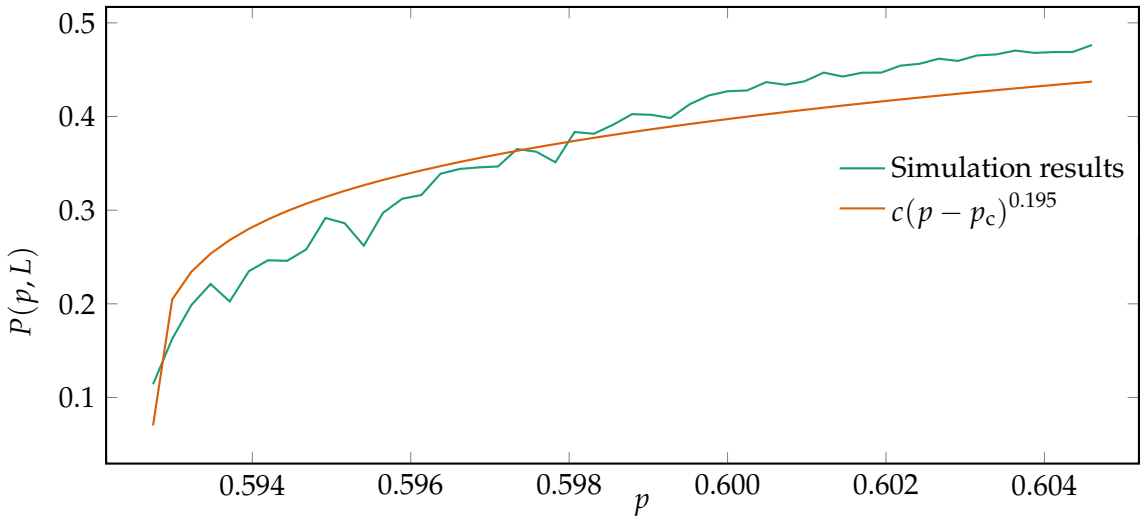


Figure 3: Spanning cluster density as a function of the probability p with $L = 1024$ and 50 cycles for probabilities in the region just above the critical probability. The result is compared with $c(p - p_c)^{\beta}$, where β is found by doing a linear fit on a logarithmic scale.

d)

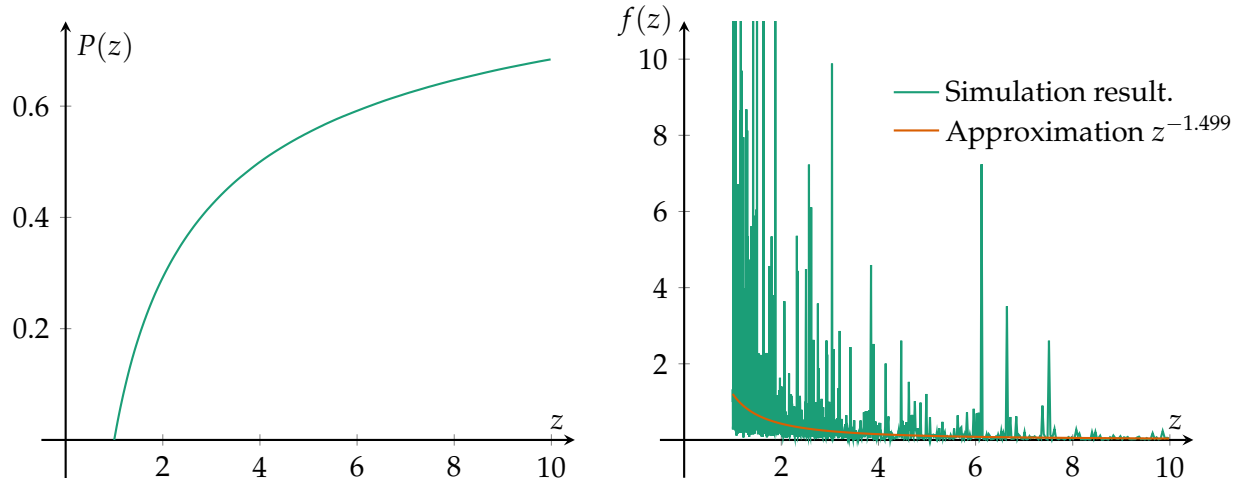
The cumulative distribution and the derivative, which approximates the probability distribution, were straightforwardly calculated using

```
z = np.random.uniform(size=N)**(-3 + 1)
z = np.sort(z)
P = np.arange(1, N + 1) / N

diff = (P[1:] - P[:-1]) / (z[1:] - z[:-1])
f[1:-1] = 0.5 * (diff[1:] + diff[:-1])
f[0] = diff[0]
f[-1] = diff[-1]

a, b = np.polyfit(np.log(z), np.log(f), deg=1)
```

and the results are shown in figure 4 on the following page.



(a) Cumulative distribution.

(b) Probability distribution.

Figure 4: The result of raising a uniform distribution between 0 and 1 to the power -2. While the cumulative distribution looks good, it is clear that some sort of averaging over bins is necessary in order to get a reasonable probability distribution.

e)

The listed algorithm can be used to calculate the cluster number density $n(s,p)$ with logarithmically binned sizes. $n(s,p)$ is defined as the probability of a randomly selected site being a specific site on a cluster with size s. It can therefore be calculated as

$$n(s,p) = \frac{\sum_{\text{cycles}} \text{number of clusters with size } s}{L^2 \cdot \text{number of cycles}}. \tag{4}$$

Due to the skewed distribution of cluster sizes, with many being small and few large clusters (except for the spanning cluster, which is ignored), this method would give similarly poor results as in figure 4. Consequently, the sizes should be binned logarithmically and the results averaged, i.e.

$$n\left([s, s+\Delta s), p\right) = \sum_{\text{cycles}} \frac{\text{number of clusters with size between } s \text{ and } s+\Delta s}{L^2 \cdot \Delta s \cdot \text{number of cycles}}. \tag{5}$$

```
loga = log(a)

num_bins  = ceiling(log(1.0d0*L**2)/loga)
bin_edges = a**[(i, i=0 ,num_bins)]
bin_mids  = 0.5*(bin_edges(1:num_bins) + bin_edges(2:num_bins+1))
bin_sizes = bin_edges(2:num_bins+1) - bin_edges(1:num_bins)

allocate(histogram(1:num_bins))
histogram = 0

!$omp  parallel do &
!$omp& private(binary_matrix, label_matrix, num_labels, &
```

```
!$omp&         clustersizes, spanning_label, sizeindex) &
!$omp& reduction(+:histogram)
do i = 1, num_samples
    binary_matrix = create_binary_matrix(p, L)
    call label(binary_matrix, label_matrix, num_labels)
    clustersizes = find_sizes(label_matrix, num_labels)
    spanning_label = find_spanning_cluster(label_matrix, num_labels)

    do j = 1, num_labels
        if(j /= spanning_label) then
            sizeindex = floor(log(1.0d0*clustersizes(j))/loga) + 1
            histogram(sizeindex) = histogram(sizeindex) + 1
        end if
    end do
end do
!$omp end parallel do

results = histogram/(L**2 * num_samples * bin_sizes)
```

f)

The cluster number density algorithm from the previous exercise was used. Figure 5 shows the result for various values of p for a fixed system size.

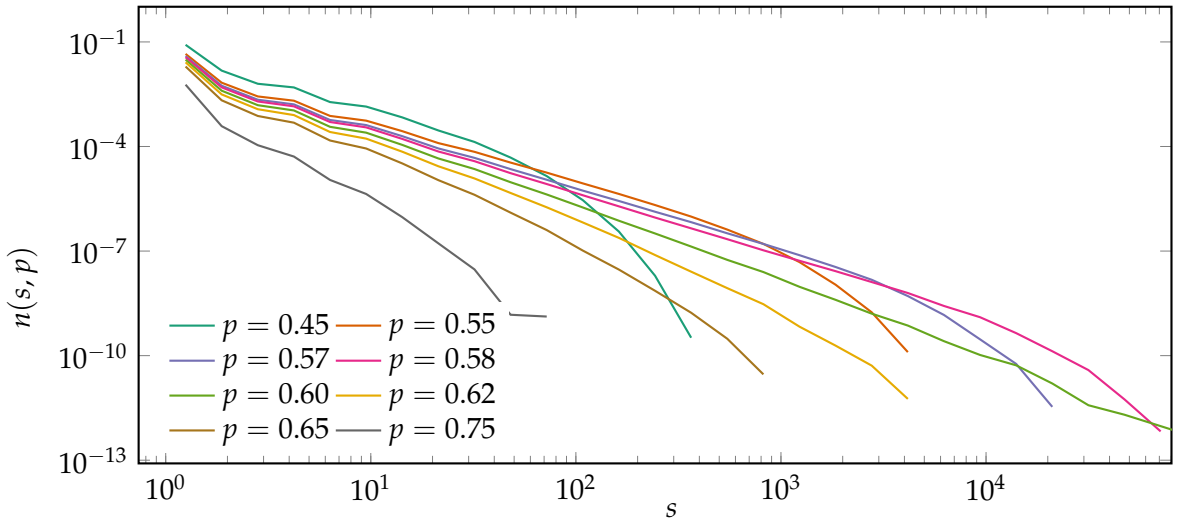


Figure 5: Cluster number density as a function of size for a few different probabilities, with $L = 1024$ and 100 Monte Carlo cycles. As p approaches p_c from below, a larger and larger portion of the graph is approximately linear. Since both axes are logarithmic, this implies a power-law behaviour, $n(s,p) \propto s^{-\tau}$.

g)

As seen in figure 5 on the previous page, the cluster number density has a large linear part when $p \to p_c$. Both axes in this figure are logarithmic, so the linearity implies a power-law behaviour for $n(s, p_c)$, i.e. $n(s, p_c) \propto s^{-\tau}$. Figure 6 shows $n(s, p_c)$ with $L = 2^i$, $i \in \{4, \ldots, 9\}$, as well as the result of a linear fit for the largest system size. The linear part is found by staring at the graph, and the resulting exponent is $\tau = 1.875$.

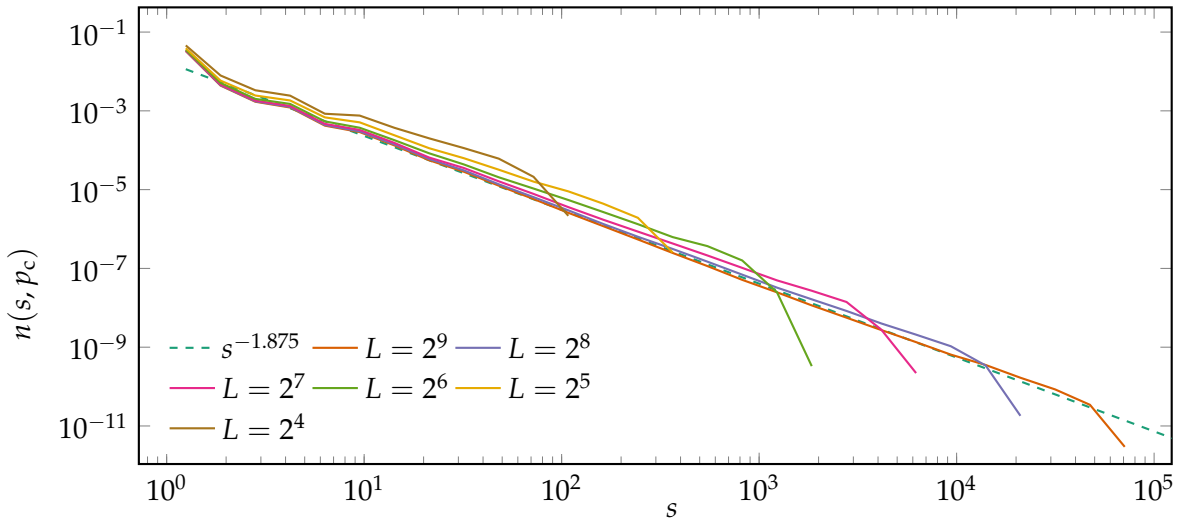


Figure 6: Cluster number density as a function of size with $p = p_c$ and 1000 Monte Carlo cycles. The power-law model fits well over a large range of sizes.

h)

Comparing figure 5 on the previous page with figure 6 shows that when $p \approx p_c$, $n(s, p)$ has a linear portion similar to that of $n(s, p_c)$, but it falls off from the linear trend when s becomes large. This point of departure can be defined as a characteristic size s_ξ. It is numerically estimated as the point where $n(s, p) \leq \frac{1}{2} n(s, p_c)$.

The characteristic size s_ξ can be assumed[7] to be proportional to $|p - p_c|^{-1/\sigma}$, where σ once again is found by doing linear regression on a logarithmic scale. Figure 7 on the next page shows the cluster number density as a function of size for probabilities approaching p_c, with s_ξ marks based on the numerical estimate described above. The resulting $s_\xi(p)$ is shown in figure 8 on the following page, and the theoretical model $|p - p_c|^{-1/0.469}$ gives a good approximation of the simulation results.

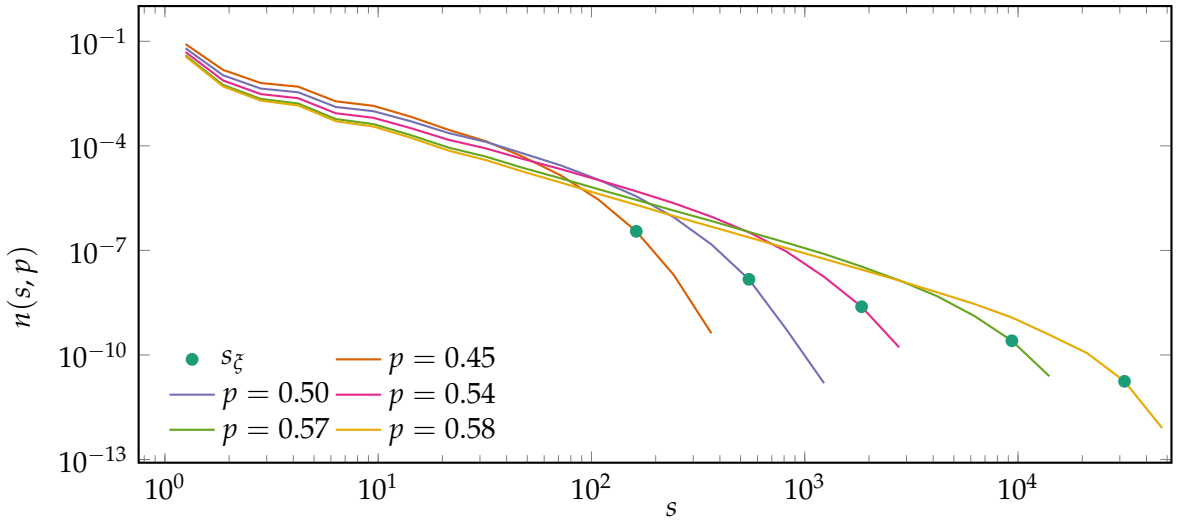


Figure 7: Cluster number density as a function of size for probabilities just below p_c with $L = 512$ and 500 Monte Carlo cycles. The dots mark the numerical estimates of s_ξ.

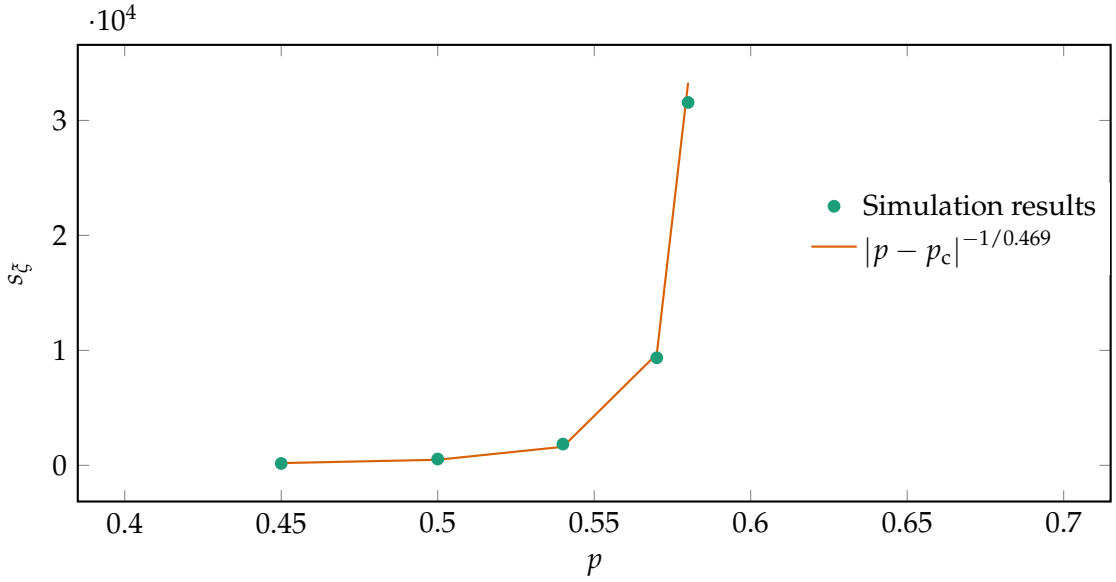


Figure 8: Characteristic cluster size s_ξ as a function of probability. As the probability approaches the critical probability, s_ξ diverges. The parameters of the power law model are found from linear regression on a logarithmic scale, and the resulting approximation is accurate.

i)

The mass of the percolating/spanning cluster M should be proportional to some power of the system size L. Intuitively, the exponent should be equal to the dimension of the system (2 in this project), but the non-trivial geometry of the spanning cluster causes the exponent to deviate from this expectation. This is confirmed by the numerically determined $D = 1.888 < 2$. Figure 9 on the next page shows the quality of the approximation.

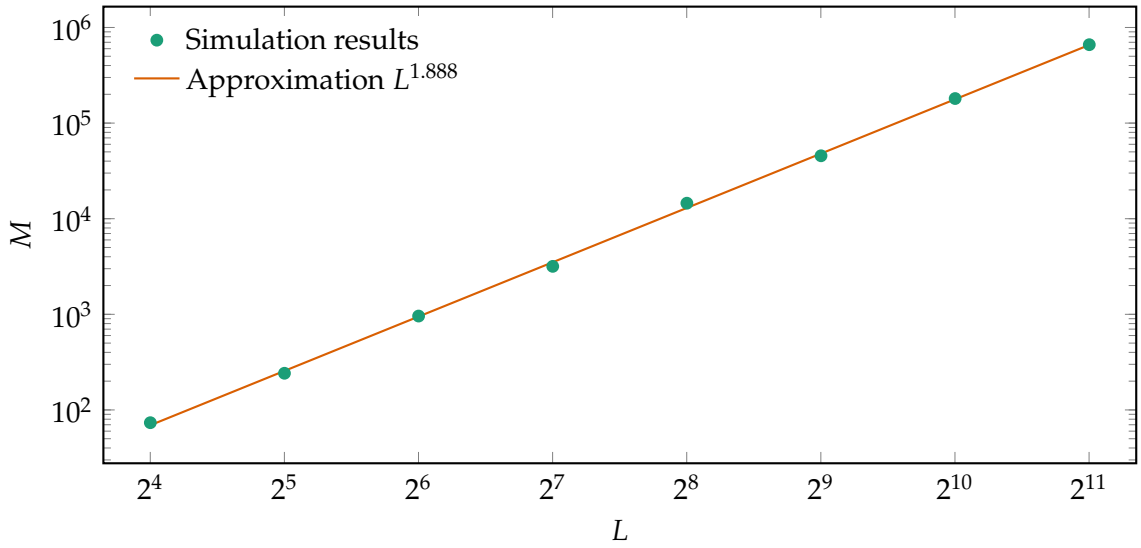


Figure 9: Mass of the percolating cluster as a function of system size, with $p = p_c$ and 200 Monte Carlo cycles. The power-law model fits well, and $D = 1.888 < 2$ shows the non-trivial geometry of the percolating cluster.

j)

Nope.

k)

Nope.

l)

In this problem we are interested in calculating the inverse of $\Pi(p, L)$, i.e. for a given number x determining $p_{\Pi=x}$ such that $\Pi(p_{\Pi=x}, L) = x$. I have chosen to implement this with the bisection method, as it is very simple but also quickly converging — the error is halved with every calculation of Π. Figure 10 on the following page shows the result.

```
lower = 0
upper = 1
lowerPI = 0
upperPI = 1

do while(upper - lower > tolerance)
    mid = (lower + upper)/2
    midPI = spanning_probability(mid, L, num_samples)
    if(midPI > x) then
        upper = mid
        upperPI = midPI
```

```
        else
            lower = mid
            lowerPI = midPI
        end if
    end do
    p_x = mid
```

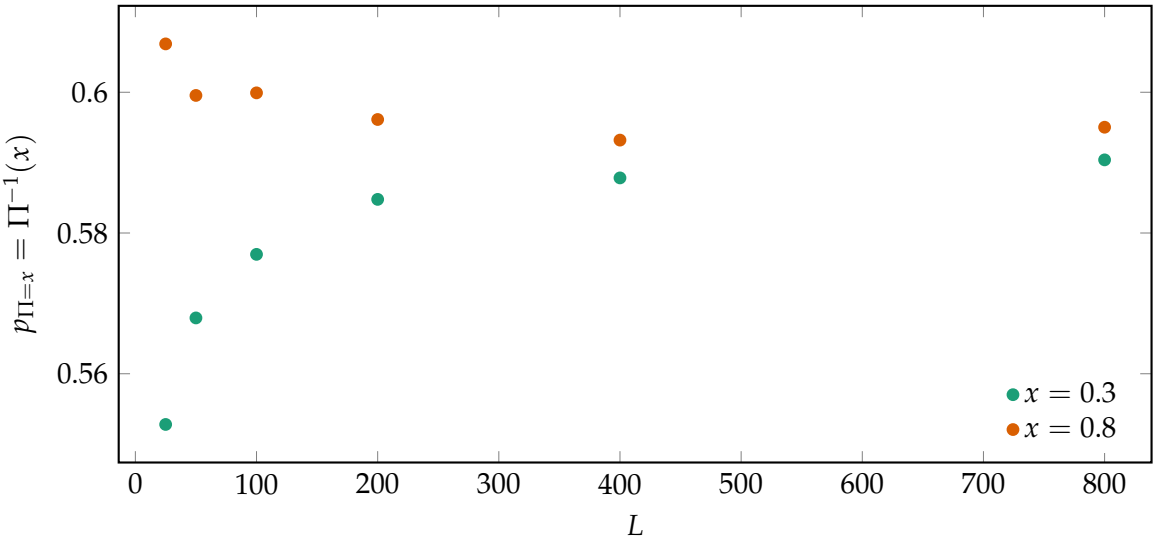
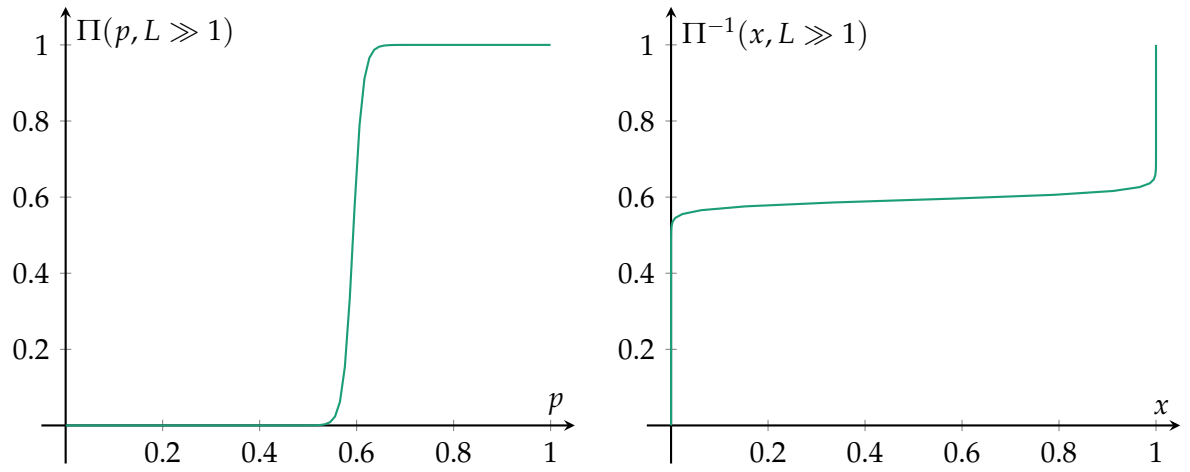


Figure 10: Inverse of the percolation probability $x = \Pi(p, L)$ as a function of L for two different values of x. As L increases, $p_{\Pi=x}$ approaches the same value for both values of x. This value must be p_c, since figure 1 on page 4 shows that $\Pi(p, L)$ approaches a step function around p_c as $L \to \infty$. $\Pi^{-1}(p, L)$ should therefore be almost constant, with value p_c, for large values of L.

m)



(a) Percolation probability.

(b) Inverse percolation probability.

Figure 11: Approximate plot of the percolation probability and its inverse when L is large. In this limit, the percolation probability is nearly a step function centred at the critical probability p_c, and the inverse is consequently approximately constant, with value p_c.

Figure 10 on the previous page shows that $\Pi^{-1}(x)$ approaches some constant value as $L \to \infty$. On the other hand, figure 1 on page 4 shows that $\Pi(p, L) \to \Theta(x - p_c)$ in the same limit, and so the constant value of $\Pi^{-1}(x)$ must be p_c. This is illustrated in figure 11.

It is difficult to exactly determine the nature of the approach to p_c from figure 10 on the preceding page, but everything else in the world of percolation is a power law, so it seems reasonable to hope that this should be the case for $\Pi^{-1}(x) - p_c$ as well, i.e.

$$\Pi^{-1}(x, L) = p_c + C(x)L^{-1/\nu} \tag{6}$$

for some ν. This equation can be used to determine ν, as

$$\Pi^{-1}(x_1, L) - \Pi^{-1}(x_2, L) = (C(x_1) - C(x_2))L^{-1/\nu},$$

so that $-1/\nu$ will be the slope when plotting $\Pi^{-1}(x_1, L) - \Pi^{-1}(x_2, L)$ as a function of L on a logarithmic scale. Linear fitting gives $\nu = 1.323$, which is close to the exact value $\nu = 4/3$. Figure 12 on the next page shows the results.

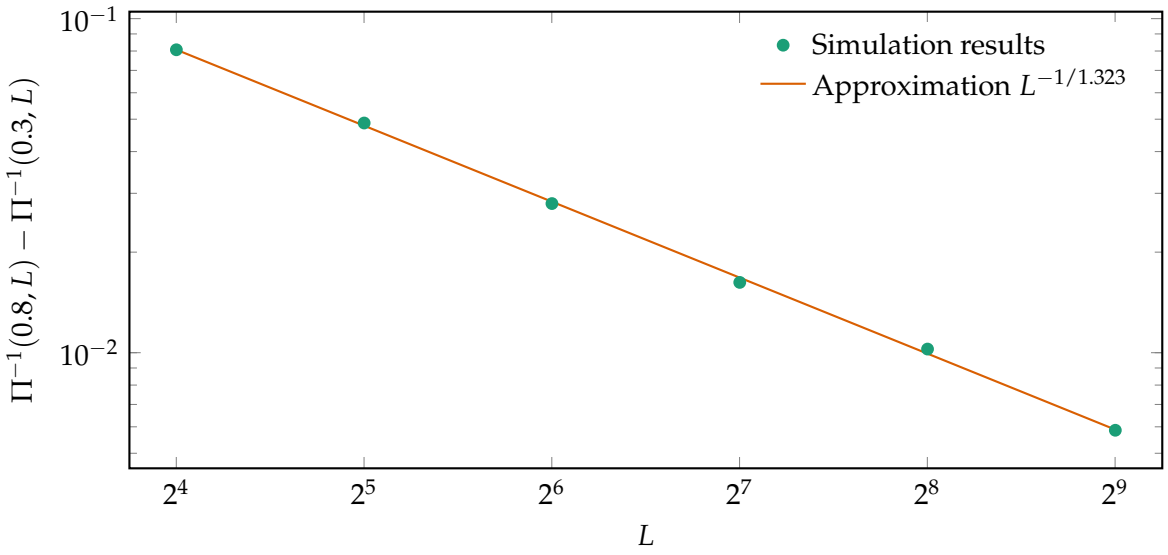


Figure 12: Difference in inverse percolation probability as a function of L. The approximately linear trend on a logarithmic scale validates the guess of yet another power-law behaviour.

n)

Figure 12 confirms the power-law behaviour of the approach of $\Pi^{-1}(x, L)$ to p_c as a function of L, i.e.

$$\Pi^{-1}(x, L) = p_c + C(x)L^{-1/\nu},$$

where ν is known to be exactly $4/3$. Π^{-1} should therefore be a linear function of $L^{-1/\nu}$, with p_c as value at $L^{-1/\nu} = 0$, i.e. $L = \infty$. The critical probability can be estimated by calculating Π^{-1} as a function of $L^{-1/\nu}$, doing linear regression and extracting the constant term. This gives figure 13, and the different values of x give similar and quite accurate estimates for p_c.

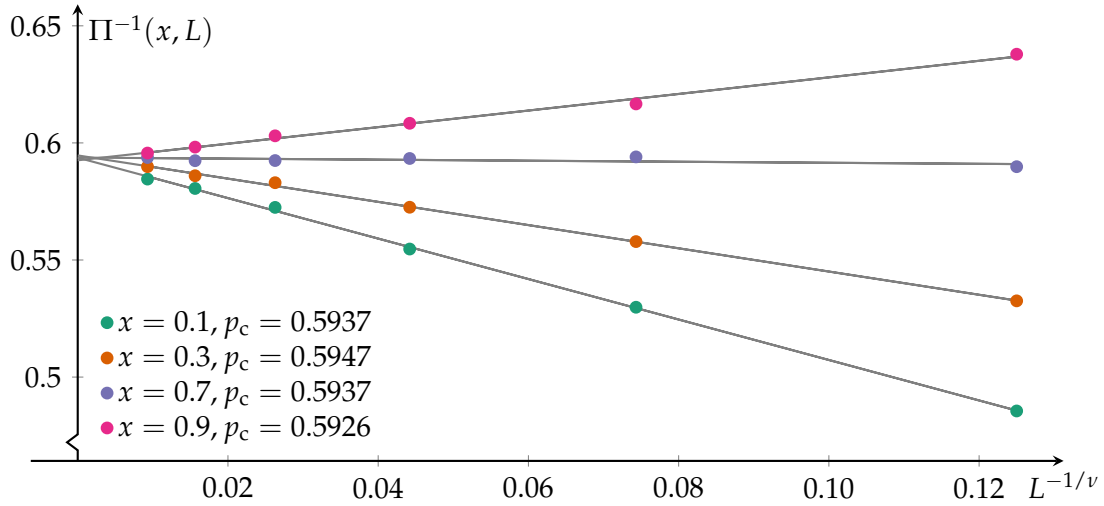


Figure 13: Inverse percolation probability $\Pi^{-1}(x, L)$ as a function of $L^{-1/\nu}$ for a few different values of x, together with linear fits. The intersection between the linear fit and the y-axis gives an estimate for the critical probability p_c.

The function $C(x)$ in

$$\Pi^{-1}(x, L) = p_c + C(x)L^{-1/\nu} \tag{7}$$

is what the lecture notes call $\Phi^{-1}(x)$. Consequently, with $p = \Pi^{-1}(x, L)$,

$$p - p_c = \Phi^{-1}(x)L^{-1/\nu} \implies \Phi^{-1}(x) = (p - p_c)L^{1/\nu}. \tag{8}$$

By definition, $x = \Pi(p, L)$, and hence applying Φ on both sides gives

$$\Pi(p, L) = \Phi\Big((p - p_c)L^{1/\nu}\Big). \tag{9}$$

The shape of the function $\Phi(u)$ can therefore be determined by plotting $\Pi(p, L)$ as a function of $(p - p_c)L^{1/\nu}$. I have chosen to vary p for a few different values of L.

Figure 14 displays the data collapse, and the results show that the graphs of $\Pi(p, L)$ overlap when $(p - p_c)L^{1/\nu}$ is used as the x-axis. The shape of the function $\Phi(u)$ looks to be what a physicist would call "reverse Fermi-Dirac with $\mu \approx 0$", i.e. a logistic function.

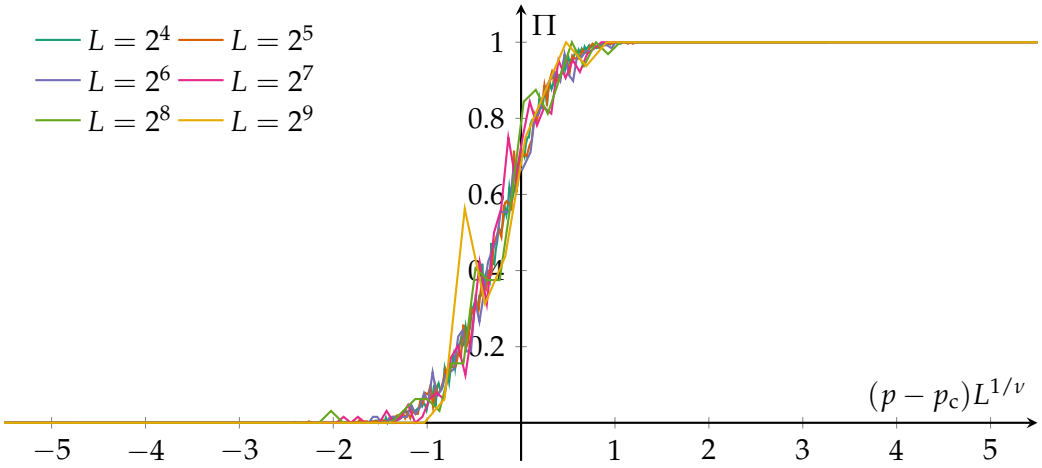


Figure 14: Percolation probability as a function of $(p - p_c)L^{1/\nu}$ for $p \in [0, 1]$ and a few different values of L. According to the finite size scaling guesses, the graphs should overlap, and the shape of the function should be that of $\Phi(u)$. The graphs do indeed overlap quite well, and the shape of $\Phi(u)$ looks to be approximately logistic.

o)

Nope.

p)

To find the singly connected bonds, i.e. the sites visited by both a right-turning and a left-turning walker, I have used the `walk` function developed by Svenn-Arne Dragly, together with the following functions:

```
from walk import walk
import numpy as np
from scipy.ndimage import measurements
import multiprocessing as mp


def M_SC_one_sample(p, L):
    binary_matrix = np.random.uniform(size=(L, L)) < p
    label_matrix, num_clusters = measurements.label(binary_matrix)

    intersect_labels = np.intersect1d(label_matrix[0, :], label_matrix[-1, :])
    percolating_cluster = intersect_labels.max()

    if percolating_cluster == 0:
        label_matrix = label_matrix.T
        intersect_labels = np.intersect1d(label_matrix[0, :],
                                          label_matrix[-1, :])
        percolating_cluster = intersect_labels.max()

    if percolating_cluster == 0:
        return 0

    percolating_matrix = label_matrix == percolating_cluster

    left, right = walk(percolating_matrix)

    return np.count_nonzero(np.logical_and(left, right))


def M_SC(p, L, num_samples):
    pLs = [(p, L)] * num_samples
    with mp.Pool(4) as pool:
        result = pool.starmap(M_SC_one_sample, pLs)

    return np.sum(result) / num_samples
```

The `walk` function implements the straightforward algorithm of left- and right-turning walkers. The right-turning walker always chooses to turn right relative to the current direction if possible, otherwise it tries to walk forward, to the left or back where it came from. Vice-versa for the left-turning walker. Once both walkers have reached the opposite side of where they came from, the sites visited by both walkers are the singly-connected bonds, i.e. the bonds whose removal would remove the percolating cluster.

q)

As with the mass of the spanning cluster, the mass of the singly connected bonds should increase as some power of the system size L when $p = p_c$. The exponent is once again found by doing linear regression on a logarithmic scale, and figure 15 on the next page shows the result together with the approximate power-law $L^{0.784}$.

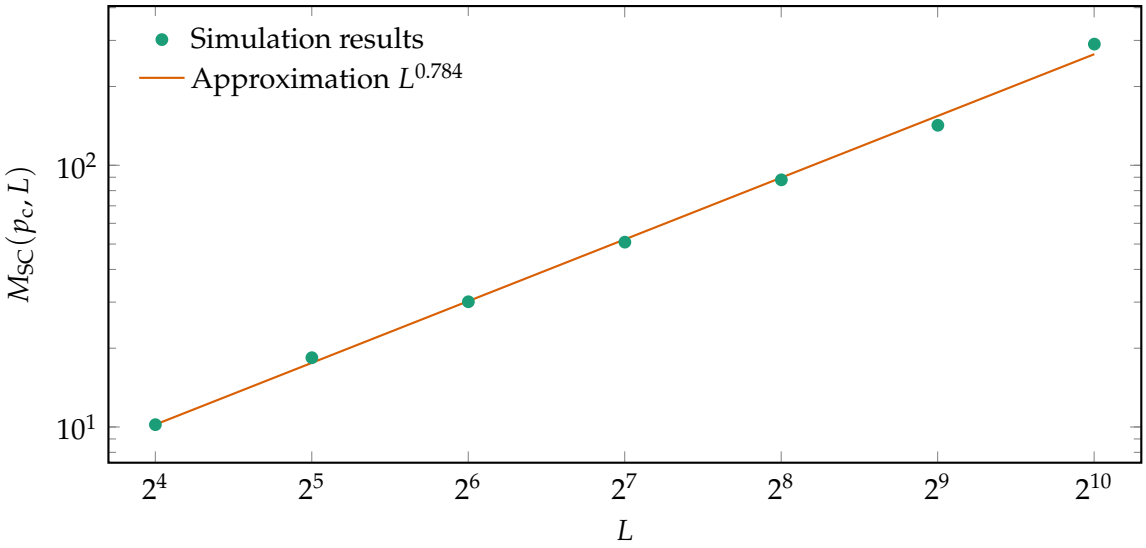


Figure 15: Mass of the singly connected bonds as a function of system size L with $p = p_c$. These quantities should be related by a power-law, which is confirmed by the linearity on a logarithmic scale.

The density of the singly connected bonds is defined in the same way as the density of the spanning cluster,

$$P_{SC}(p, L) = \frac{M_{SC}(p, L)}{L^2}.$$

This is largest at p_c, as there will then (on average) barely be a spanning cluster, and hence many singly connected bonds. When $p < p_c$ it is not guaranteed that there is a spanning cluster at all, while $p > p_c$ increases the probability of having multiple routes from one side to the other, thus reducing the number of singly connected bonds.

The described behaviour is similar to that of $s_{\tilde{\zeta}}$, which also diverged for $p \to p_c$. This quantity was found to be proportional to a negative power of $|p - p_c|$. According to multiple sources[2, 8] the density of the singly connected bonds should have a similar relation,

$$P_{SC}(p, L) \propto |p - p_c|^{-1},$$

independent of both lattice and dimensionality. Figure 16 on the following page shows fairly good fit between a power-law and the simulation results, with

$$P_{SC}(p, L) \propto |p - p_c|^{-1.026}.$$

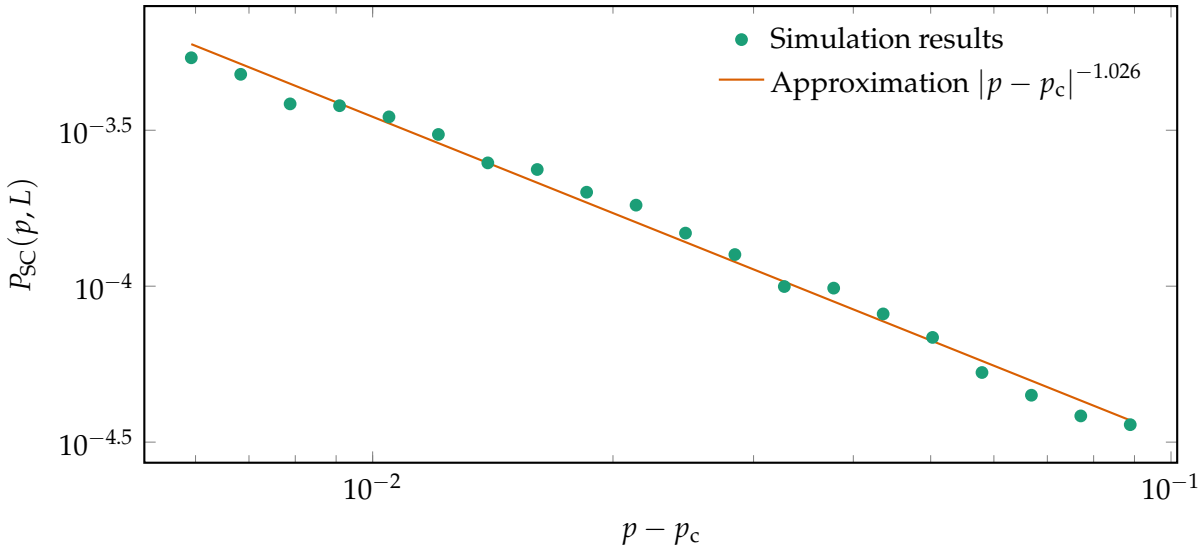


Figure 16: Density of the singly connected bonds as a function of probability in an interval around the critical probability with $L = 512$.

References

[1] Walter S. Brainerd. *Guide to Fortran 2008 programming*. London: Springer, 2015. 408 pp. ISBN: 978-1-4471-6758-7.

[2] Antonio Coniglio. "Cluster structure near the percolation threshold". In: *Journal of Physics A: Mathematical and General* 15.12 (1st Dec. 1982), pp. 3829–3844. ISSN: 0305-4470, 1361-6447. DOI: 10.1088/0305-4470/15/12/032. URL: http://stacks.iop.org/0305-4470/15/i=12/a=032?key=crossref.cb6432c859595b28dcaefaaa6ecb2c43 (visited on 15/05/2018).

[3] Tobin Fricke. *The Hoshen-Kopelman Algorithm*. 21st Apr. 2004. URL: https://www.ocf.berkeley.edu/~fricke/projects/hoshenkopelman/hoshenkopelman.html (visited on 06/05/2018).

[4] Lifeng He, Yuyan Chao and Kenji Suzuki. "A Run-Based Two-Scan Labeling Algorithm". In: *IEEE Transactions on Image Processing* 17.5 (May 2008), pp. 749–756. ISSN: 1057-7149. DOI: 10.1109/TIP.2008.919369.

[5] Joseph Hoshen and Raoul Kopelman. "Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm". In: *Physical Review B* 14.8 (15th Oct. 1976), pp. 3438–3445. DOI: 10.1103/PhysRevB.14.3438. URL: https://link.aps.org/doi/10.1103/PhysRevB.14.3438 (visited on 06/05/2018).

[6] Yacov Kantor. "Geometrical properties of single connected bonds in percolation clusters". In: *Journal of Physics A: Mathematical and General* 17.15 (1984), p. L843. ISSN: 0305-4470. DOI: 10.1088/0305-4470/17/15/005. URL: http://stacks.iop.org/0305-4470/17/i=15/a=005 (visited on 15/05/2018).

[7] Anders Malthe-Sørenssen. *Percolation and Disordered Systems A Numerical Approach*. 30th Apr. 2015.

[8] Abbas Ali Saberi. "Recent advances in percolation theory and its applications". In: *Physics Reports* 578 (May 2015), pp. 1–32. ISSN: 03701573. DOI: 10.1016/j.physrep.2015.03.003. arXiv: 1504.02898. URL: http://arxiv.org/abs/1504.02898 (visited on 15/05/2018).

[9] Dietrich Stauffer, Antonio Coniglio and Mireille Adam. "Gelation and critical phenomena". In: *Polymer Networks*. Ed. by Karel Duek. Vol. 44. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 103–158. ISBN: 978-3-540-11471-0. DOI: 10.1007/3-540-11471-8_4. URL: http://link.springer.com/10.1007/3-540-11471-8_4 (visited on 15/05/2018).