FYS4460 Project 3

Svenn-Arne Dragly

In this project we are working with percolation, the study of connectivity and other properties in random media.

I. INTRODUCTION

Percolation is the study of connectivity and other properties in random media. We will in this project study a two-dimensional system where a binary matrix is chosen at random with a given probability p for each of its sites to be occupied or not. These sites will be grouped into clusters which are studied for their connectivity, areas, masses and other properties. In this project, we will dig into topics such as percolation treshold and power law distributions.

II. THEORY

A. Definitions

For the reference (and to give myself somewhere to look up stuff), I've listed the definitions used throughout this project in table I.

Term	Definition
cluster	Set of sites that are connected
connected	Sites that share sides. In this project, we define connections with 4 neighbors on a square lattice
spanning cluster	A cluster that spans the whole system from edge to edge in either direction
finit cluster	Any cluster that is not a spanning cluster

TABLE I. Definitions used throughout this project

There are also a set of properties that we will explore, such as the probability for a site to be part of the spanning cluster and the probability for a spanning cluster to exist at all. These are listed in table II.

Property	Definition
p	Probability that a site will be occupied.
$P(p, L)$	Probability that a site is part of the spanning cluster
$\Pi(p, L)$	Probability that a spanning cluster exists in the system
$g(r, p, L)$	Probability that two sites a distance r apart belong to the same spanning cluster
s	Denotes the size of a cluster
$\xi(p, L)$	Correlation length
$sn(s, p, L)$	Probability for a site to belong to a cluster of size s.
$n(s, p, L)$	Cluster density - a useful property that among other things is used to find $sn(s, p, L)$

TABLE II. Properties that will be explored in this project

III. IMPLEMENTATION

Python has been used as the main programming language for this project. To do this, I needed to find the relevant functions to use in Python that corresponds to the listed functions in MATLAB.

A. Working with percolation clusters in Python

For original article, see this webpage.

First of all, we need to include the pylab package together with the scipy.ndimage.measurements package:

```
from pylab import *
from scipy.ndimage import measurements
```

Now, let's create a random matrix of filled and unfilled regions and visualize it:

```
L = 100
r = rand(L,L)
p = 0.4
z = r<p

imshow(z, origin='lower')
colorbar()
title("Matrix")
show()
```

This creates a matrix of random numbers r before creating a map z of this matrix where each element is set to True if $r < p$ and False if $r > p$ as shown in figure 1.

Furthermore we want to label each cluster. This is performed by the scipy.ndimage.measurements.label function:

```
lw, num = measurements.label(z)
imshow(lw, origin='lower')
colorbar()
title("Labeled clusters")
```

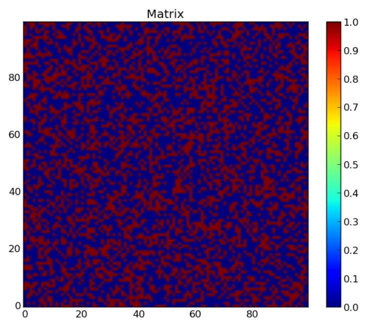


FIG. 1. Example of a binary matrix that we'll perform our calculations on.

```
show()
```

This labels each cluster with a number which may be visualized using the imshow function as shown in figure 2.

Beacause the label starts from the bottom up, the cluster colors are a bit too ordered, making it hard to distinguish two clusters close to each other. To fix this, we may shuffle the labeling with the following code:

```
b = arange(lw.max() + 1)
shuffle(b)
shuffledLw = b[lw]
imshow(shuffledLw, origin='lower')
colorbar()
title("Labeled clusters")
show()
```

Now it is way easier to see the different clusters as shown in figure 3.

After this we may want to extract some properties of the clusters, such as the area. This is possible using the scipy.ndimage.measurements.sum function:

```
area = measurements.sum(z, lw,
     index=arange(lw.max() + 1))
```

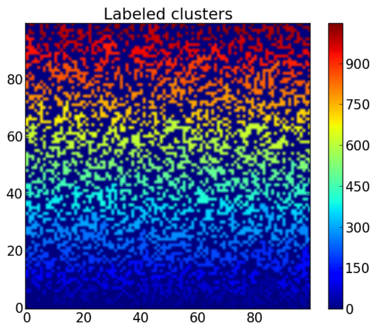


FIG. 2. Matrix labels, with colors ordered from bottom to top.

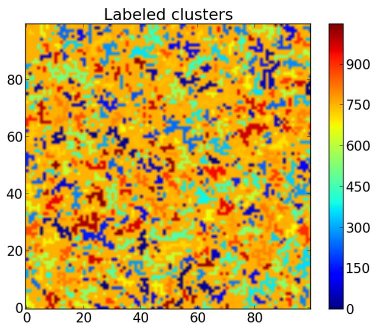


FIG. 3. Matrix with labels in random order.

```
areaImg = area[lw]
im3 = imshow(areaImg, origin='lower')
colorbar()
title("Clusters by area")
show()
```

The above code now plots the same matrix as above, but this time with all clusters colored by area.

Finally, we may want to find the bounding box of the largest cluster, so we again may see if there is a path from one side to the other. This is possible by using the function scipy.ndimage.measurements.find_objects. Note however that this is a bit risky. If there are two clusters that are largest with the same area, find_objects will find the bounding box of both clusters. That's why we'll need to rather loop over the labels for the clusters with max areas. This is done by the following code:

```
labelList = arange(lw.max + 1)
maxLabels = labelList[where(area == area.max())]
for label in maxLabels:
  ...
```

In this for loop, we use the find_objects function to plot a rectangle around the largest clusters:

```
sliced = measurements.find_objects(lw == label)
if(len(sliced) > 0):
    sliceX = sliced[0][1]
    sliceY = sliced[0][0]
    plotxlim=im3.axes.get_xlim()
    plotylim=im3.axes.get_ylim()
    plot([sliceX.start, sliceX.start, sliceX.stop,
         sliceX.stop, sliceX.start], \
                     [sliceY.start, sliceY.stop,
                          sliceY.stop,
                          sliceY.start,
                          sliceY.start], \
                     color="red")
    xlim(plotxlim)
    ylim(plotylim)

show()
```

This final plot is shown in figure 4 and the whole script is located in the Git repository for this project.

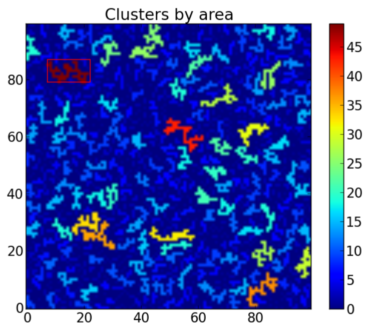


FIG. 4. Matrix with clusters colored by area.

B. Probability for a site to be part of the spanning cluster

To find the probability for a site to be part of the spanning cluster, $P(p, L)$, I've iterated over different p in the range $p = 0.5$ to $p = 1$ (by inspection, there was no reason to include values for $p < 0.5$). For each p, the above program is run 1000 times while the **sliceX** and **sliceY** variables are checked against the size of the whole matrix:

```
if sliceX.stop − sliceX.start >= Lx or sliceY.stop
     − sliceY.start >= Ly:
    P = area.max() / (Lx ∗ Ly)
else:
    P = 0
```

We then find the average of all P for each p. In principle this could easily be extended to systems of rectangular shape. To extend it further to systems with for instance hexagonal boundaries or shearing, further steps would be needed to compare the edges of the boundaries to the spanning cluster. This is likely just a matter of some geometric operations, for instance creating bounding boxes near the boundaries and seeing if these intersect the edge or not. This has not been attempted in this project.

For the case with $L = 100$, P goes as a function of p as shown in figure 5.

C. The form of the function

We now have an estimate for the probability for a site to be part of the spanning cluster for different values of p. From here we may want to see how well this fits the theory which says that for $p > p_c$, the probability $P(p, L)$ for a given site to belong to the percolation cluster will be on the form

$$P(p, L) \sim (p - p_c)^{\beta}. \tag{1}$$

To put this to the test, we take the logarithm on both sides to find

$$\ln P(p, L) \sim \beta(p - p_c). \tag{2}$$

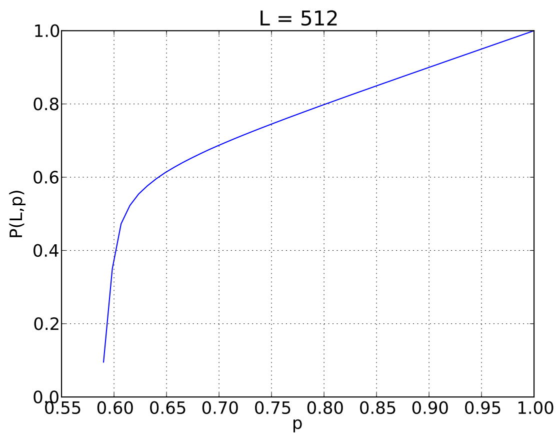


FIG. 5. Probability for a site to be part of the spanning cluster $P(p, L)$ plotted as a function of p. This was found by sampling P for 500 random binary matrices for different values of p.

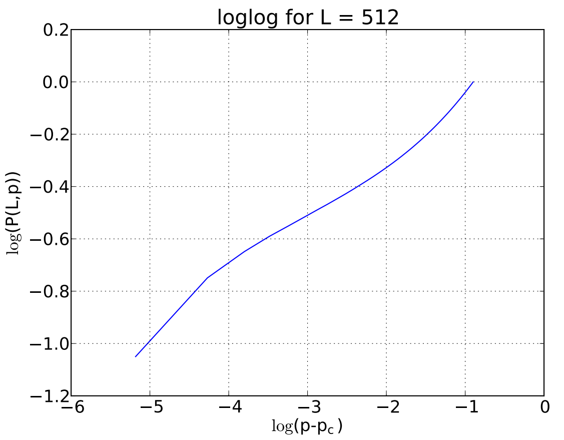


FIG. 6. loglog plot of the probability for a site to be part of the spanning cluster $\log P(p, L)$ plotted as a function of $\log(p - p_c)$. This was found by sampling P for 500 random binary matrices for different values of p.

Meaning that we should be able to read off the gradient in a loglog plot of $\log P(p, L)$ vs $\log(p - p_c)$. Such a plot is shown in figure 6. By taking the derivative of this function (numerically) we get the plot shown in figure 7. From this plot we may read off the lowest value of the derivative as an estimate for β. In this case it appears to be about $\beta = 0.17$. This is a bit higher than what is the expected value of $\beta = 0.14$.

D. Power-law distributions

Before addressing cluster number densities, we need some tools to help us determine the exponent of power-law distributions. To do this, we are creating a set of random data points in Python:

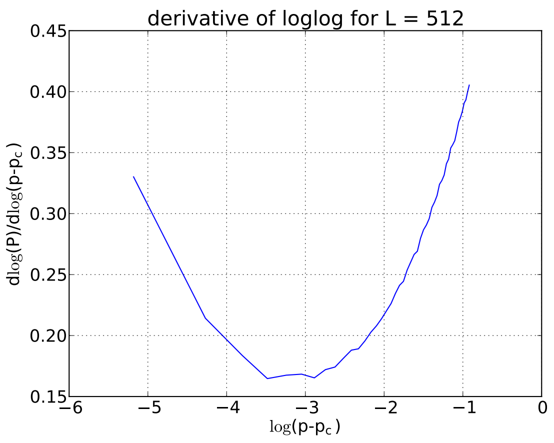


FIG. 7. Derivative of the loglog plot of the probability for a site to be part of the spanning cluster $\log P(p, L)$ plotted as a function of $\log(p - p_c)$. This was found by sampling P for 500 random binary matrices for different values of p.

```
z = rand(1e6)**(-3+1)
```

This distribution is not suited for a regular histogram, but one with logarithmic binning will do. We may therefore construct a series of bin edges determined by Python's **logscale** function:

```
mybins = histogram(z,
     bins=logspace(0,log10(z.max()),100))
```

We may now calculate the cumulative distribution function with data points within this binning by using Python's **cumsum** function and divide by the sum of all bins to find $P(Z > z)$:

```
cumulativeDistribution = cumsum(mybins[0]) /
     float(sum(mybins[0]))
```

The result is shown in figure 8. Taking the derivative of the cumulative distribution gives us the actual distribution

$$f_Z(z) = \frac{dP(Z > z)}{dz} \tag{3}$$

After calculating this, we take the logarithm of both $f_Z(z)$ and z and plot these against each other in figure 9. This shows us that we clearly have a distribution on the form $f_Z(z) \propto z^{\alpha}$ where α is the incline of the curve in figure 9. To find this exactly, we may take the numeric derivative and plot it as shown in figure 10. Here we clearly see that the value for α is approximately $\alpha = -0.5$.

With these tools in place, we can move on to calculating the cluster number density.

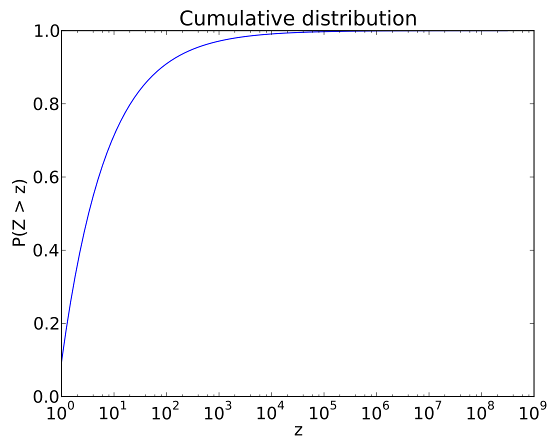


FIG. 8. Cumulative distribution of the randomly selected data points.

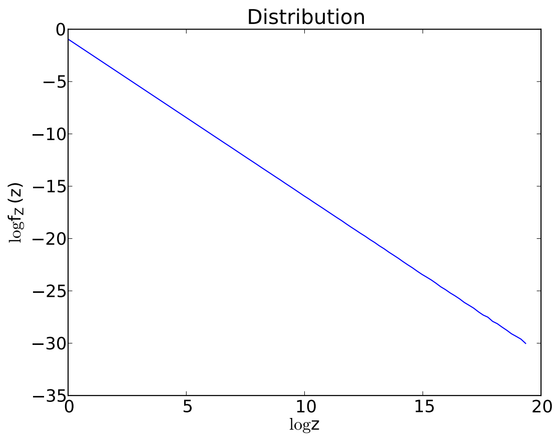


FIG. 9. Cumulative distribution of the randomly selected data points.

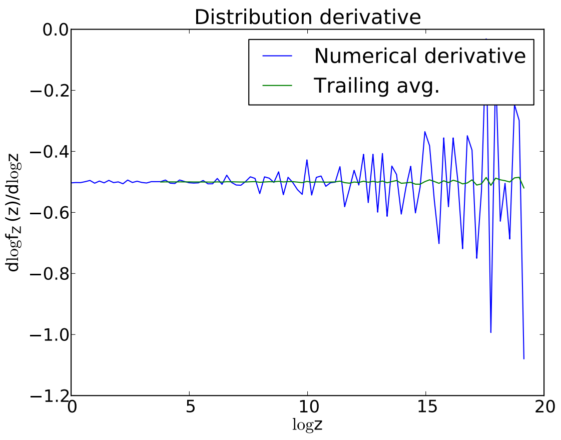


FIG. 10. Cumulative distribution of the randomly selected data points.

E. Cluster number density

The cluster number density $n(s,p)$ can be calculated by running the program we used in the first exercises to find the area for each cluster. Thereafter, we may find the number of occurences each area and weight each occurence by the area itself using the NumPy histogram function:

```
currentBins = histogram(area, bins=bins,
     weights=area)[0]
```

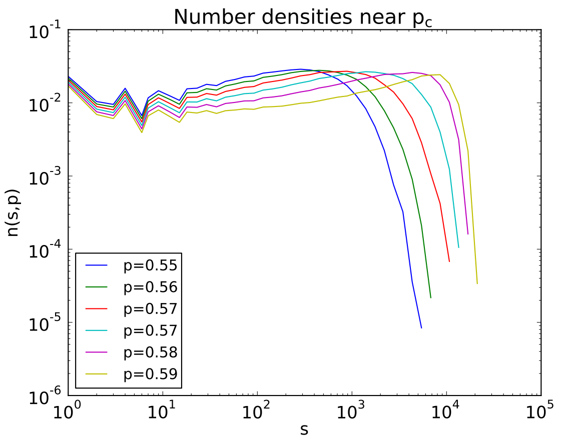
Before doing this, we set the area of the spanning cluster to zero, just to make sure it does not contribute to the statistics.

If we now divide by the total area of the system, we will find the cluster number density for this specific system.

It is interesting to see what happens to the cluster number density as p approaches p_c from above and below. This is shown in figure 11.

Furthermore, we might be interested in seeing how the cluster number density $n(s,p)$ as a function of s varies with the system size L. This is shown in figure 12

This allows us to estimate τ.



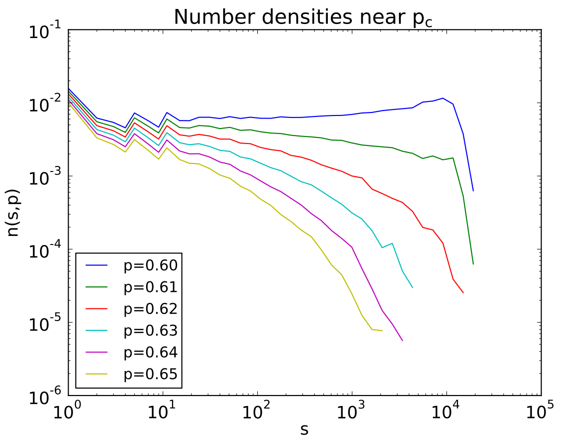


FIG. 11. loglog plot of the cluster number density for a system with $L = 256$ using 1000 samples for each value of p as we approach p_c from below (left) and above (right).

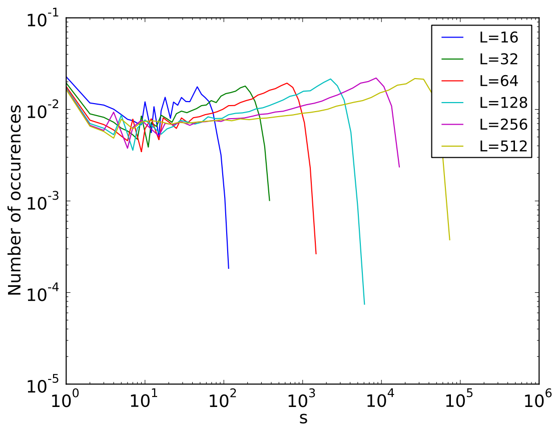


FIG. 12. Cluster number density $n(s, p_c)$ as function of s for different system sizes.