Percolation

A powerful tool in simulation

by Ipshita Bonhi

on December 7, 2020

What is it?

Consider a sponge as scene at the right. This kind of medium have tiny pores inside a solid skeleton. These kind of mediums are called “Porous Medium”.

Now suppose we have to measure the water flow through such materials. Since the pores inside a porous medium are placed completely randomly.

So in order to study such random porous objects, we need a simulation method called Percolation, which gives us to build a discrete version of these materials and study the connnected regions inside the medium.

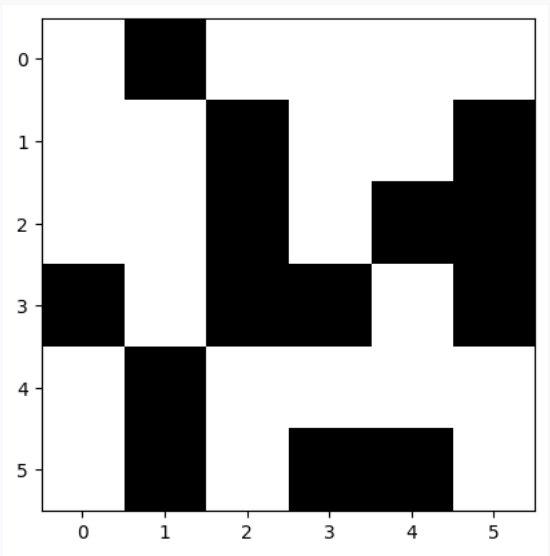
This model also gives us the ability to monitor the spacous parts as well. It is most useful for physists and applied mathematicians. It also has applications to biology, computer science, and social sciences.

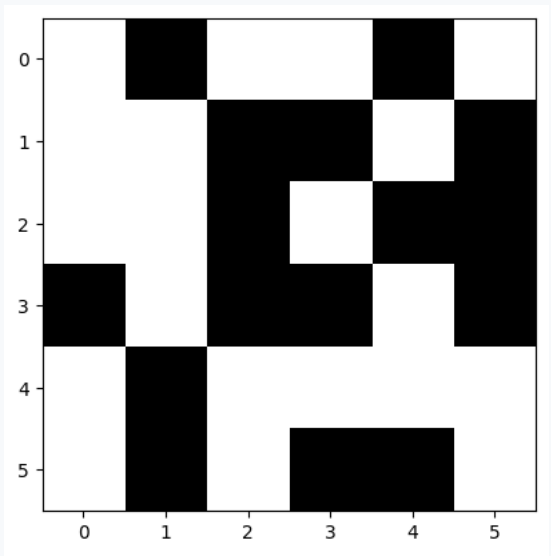
In a percolation model, we consider an $L \times L$ box, which we call lattice.

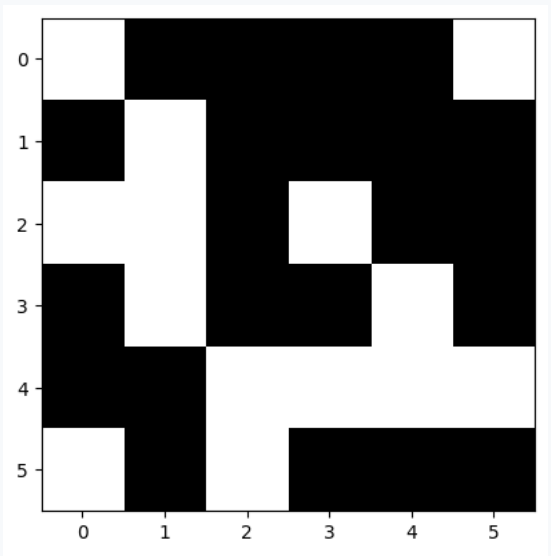
We toss a biased coin with probability p for heads and probabilty $1 - p$ for tails to fill some of the cells up.

So for each of the cells, we toss a coin. If the coin lands on heads, then we color the cell black, which means it is filled, and otherwise keep it empty.

For example, the following three 6×6 lattices are randomly filled with probabilities $p = .4, .5, .6$ respectively



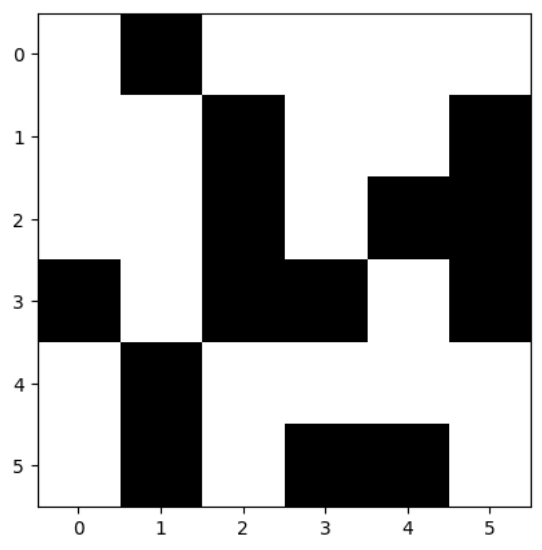


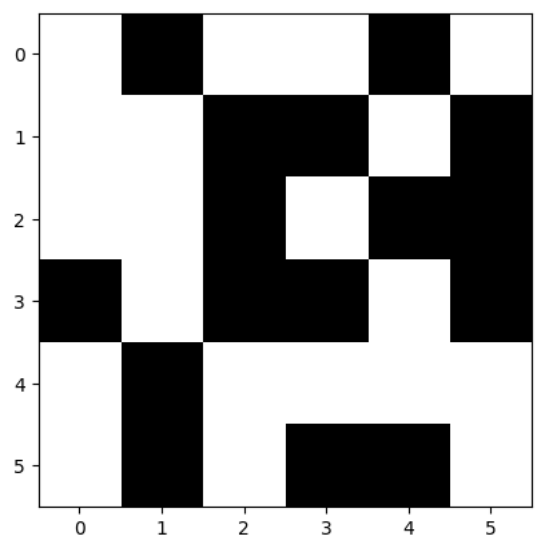


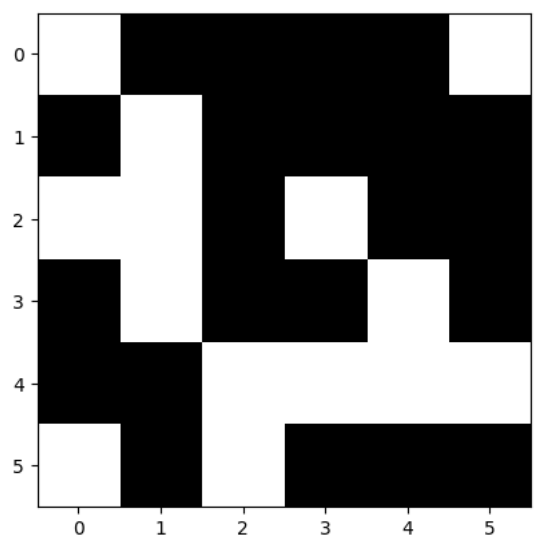
Percolation is the study of connectivity.

In such a randomly generated lattice, we are interested in connected clusters. A connected cluster is made of some cells that are neighbor to each other and are all occupied.

So in the first picture below, there are 6 connected clusters and there are $7,$ and 4 in the second and third picture.



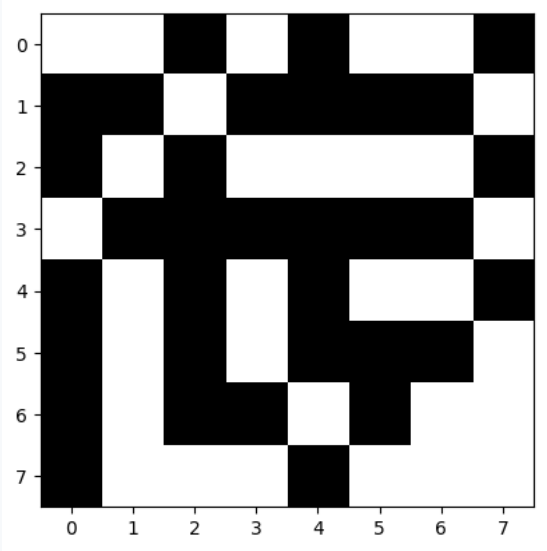


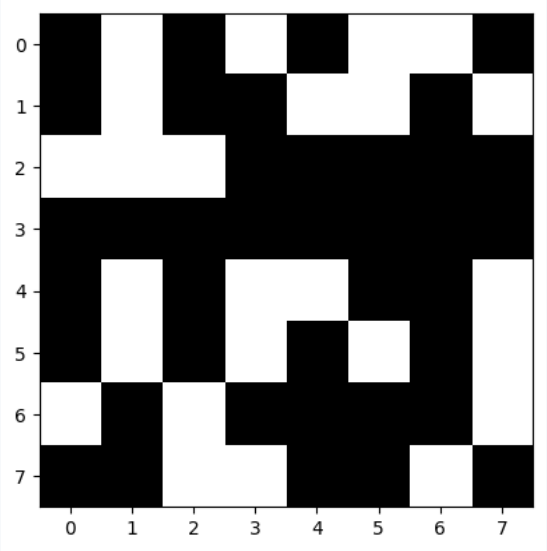


We are even more interested in connected clusters that spans from one side of the lattice to the other. We call them Spanning Components.

We can such systems where there is a spanning cluster Percolating system.

For example, there is no spanning cluster that goes from top to bottom of the lattice in the first lattice below, but there is one in the second lattice.





In short, we will ask this question:

If we randomly generate a lattice with a biased coin with probability p, for which values of $0 \leq p \leq 1$ at system is almost certainly to be **Percolating**?

In other words, when will we almost certainly find a spanning connected cluster?

And in those scenarios, what happens to the connected clusters?

Before we jump in to analyze percolation models and when such models are percolating, let's take a look at why do we even care about percolating theory beside studying porous materials.

- In situations where objects are linked to each other, and their properties effect others connected to them. For example in social marketing scenarios where buyers influence each others' preferences.
- In biological sciences, to predict the fragmentation of biological virus shells.
- In ecology to see how environment fragmentation impacts animal habitats.
- In multilayerd computer networking to find out how the layers interact with each other.
- To study traffic system in a city. Dynamic percolation models can predict the traffic capacity thresholds of particular junctions or roads.
- It has also found its application in discrete mathematics such as graph theory and different graph algorithms.

and many more...

Taking a Closer Look

* **Basic structure**
* **Defining Percolation Threshold**
* **Finding the Percolation Threshold**

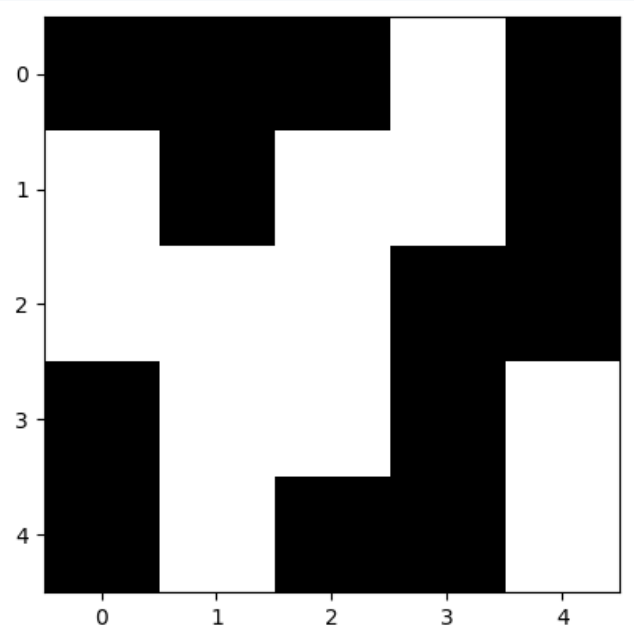
Taking a Closer Look

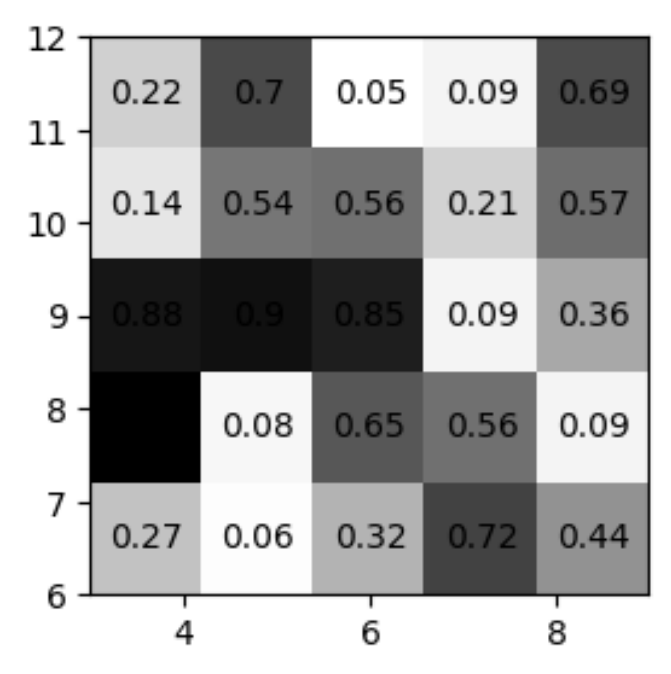
* **Basic structure**
* **Defining Percolation Threshold**
* **Finding the Percolation Threshold**

So, how do we write a very basic version of this model? We want to create a lattice, generate a random number between $0, 1$ for each of the cells, and check with the probability p if that cell should be occupied or not. The following simple code does such that.

```
1  import matplotlib.pyplot as plt
2  import numpy.random as rnd
3
4  p = 0.5              # defining the probability here
5  l = rnd.rand(5, 5) # create a 8x8 array with a random probability
6  m = l < p            # checking if cells are occupied, true or false
7
8  plt.imshow(m, interpolation='None', cmap='binary')
9  plt.show()
```

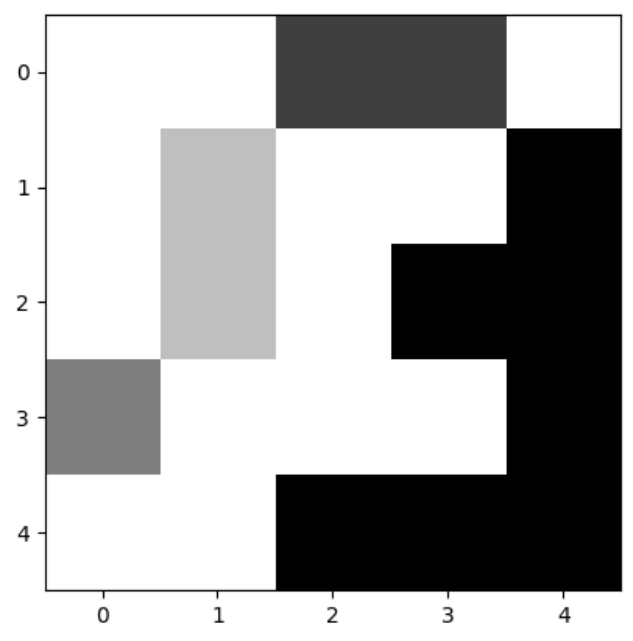
For example, one instance of this program would create the first figure shown below. On the second figure you can see the values of each random probability for each of the cells.





Now since we are interested about the clusters, we can have them in different shades to be like:

```
1  from scipy.ndimage import measurements
2  from random import shuffle
3
4  lw, num = measurements.label(m)
5  b = np.arange(lw.max() + 1)
6  shuffle(b[1:])
7  shuffledLw = b[lw]
```



Taking a Closer Look

* Basic structure
* **Defining Percolation Threshold**
* Finding the Percolation Threshold

We previously said that we will be interested at when we have a spanning cluster. To get a better understanding, we define the following:

Percolation Threshold: The smallest probability p for which any randomly generated lattice of a fixed size with probability p is almost certain to have a spanning cluster.

For example, the percolation threshold for a 2D infinite lattice is 0.593 meaning that if we color the cells of an infinite 2D lattice with the probability $p = 0.593$, then there will be certainly a cluster that spans from negative infinity to positive infinity.

Now the natural question is, why is the threshold so important?

Say we have a random situation, where our product quality has to be a certain level to reach the most number of customers. Now we want to minimize the cost. So if we consider the situation to be a percolation model, our quality should be somewhere close to the percolation threshold.

Taking a Closer Look

* Basic structure
* Defining Percolation Threshold
* **Finding the Percolation Threshold**

Now, how do we find the percolation threshold for a given lattice of size $L \times L$?

Previously where we used `lw, num = measurements.label(m)`, we named the clusters by numbers. For example, this like would turn the left `nparray` to the right `nparray`.

```
[[ True  True False  True False]
 [ True False  True  True False]
 [False  True False False False]
 [ True  True False  True False]
 [ True False False  True False]]
```

```
[[1 1 0 2 0]
 [1 0 2 2 0]
 [0 3 0 0 0]
 [3 3 0 4 0]
 [3 0 0 4 0]]
```

So if a spanning cluster existed, then the leftmost column and the rightmost column would have one number in common, which would mean that there were a cluster that spanned from the left border to the rightmost border.

To find the percolation threshold, we divide the interval $[0, 1]$ into 100 or more equally spaced points. And then run the simulation to randomly generate lattices and check if there is spanning cluster in that lattice. The pseudocode is given below:

```
arr = np.linspace(0, 1, 100)
for p in arr:
    samples = produce_samples(p)
    if check_if_spanning_cluster_exists(samples):
        return p
```

This will find the smallest value of p for which almost certainly random lattices will be percolating.