Percolation Threshold of 2D Disk Percolation

Micol Bedarida Jan. 11th 2015

Abstract

The percolation threshold for two-dimensional continuum disk percolation was calculated in two distinct ways using simulations in the programming language Python 2.7. The results yield area density percolation thresholds of 1.14 and approximately 1.17, both of which lie within 4% of the currently accepted value.

Introduction

Percolation theory describes the behaviour of connected clusters in a geometric lattice or in a random disposition of geometric objects. This theory was formulated in 1957 by Broadbent and Hammersley to model the flow of fluid in porous mediums[1]. It has since evolved into a vast field which involves difficult problems and which is useful for analysing and predicting phenomena such as forest fires and epidemics. Percolation theory is usually subdivided into discrete percolation models, which describe discrete objects in a lattice, and continuum percolation models, which describe randomly distributed geometric objects. Research in this area has for the most part focused on lattice percolation even though most natural systems correspond to continuum percolation[2]. This project analyses the behaviour of two-dimensional continuum disk

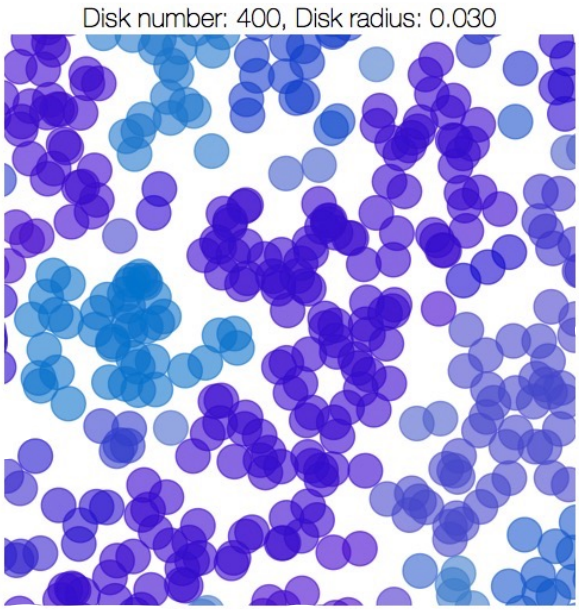


Figure 1: Non-percolating system. Each cluster is highlighted in a different shade of blue.

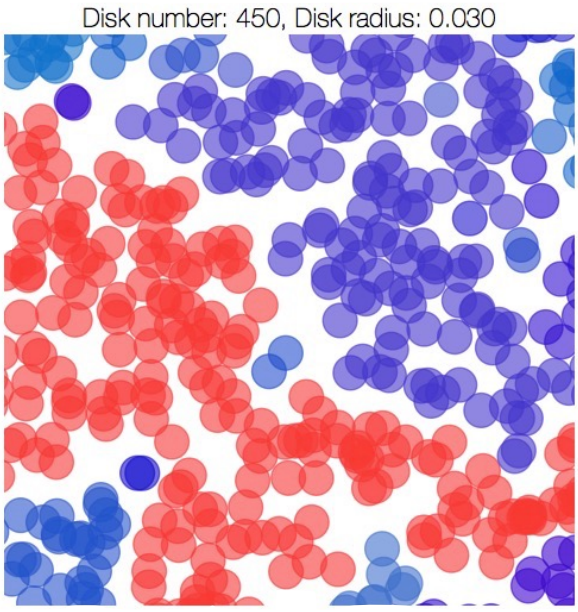


Figure *2*: Percolating system. The percolating cluster is highlighted in red.

[1] Bollobas B, Riordan O. *Percolation*. Cambridge: Cambridge University Press. 2006. (Page 1)

[2] Martens S, Moore C. *Continuum Percolation Thresholds in Two Dimensions*. 2012. Available from: http://arxiv.org/pdf/1209.4936.pdf. (Page 1)

percolation inside a unit square. Percolation is defined as taking place when a cluster of overlapping disks spans the square, thereby connecting the left side to the right side. To give an example, the system in *Figure 1* does not percolate whereas the system in *Figure 2* percolates because the cluster of disks shown in red spans the square horizontally.

Computational method

Code structure

The programming language Python 2.7 was used to simulate and analyse random dispositions of disks with constant radius inside the unit square.

The following functions were defined in order to partition the disks into clusters. The function `check_overlap` computes the disks which overlap a given disk. Another function `join_clusters` combines two clusters into one when a disk spans both clusters. The function `find_clusters` uses the previous two functions to identify all of the clusters.

The function `find_percolation` calls `find_clusters` after generating random disk coordinates and is used by three top-level functions. The first top-level function, `single_run`, was used interactively for exploration purposes, whereas the functions `multiple_runs_fixed_radius` and `multiple_runs_fixed_density` were used to perform repeated batch-mode simulations, iterating through the experiment under the conditions of fixed radius and fixed density respectively. This was done to explore the probability of percolation under the condition that the radius is fixed as the number of disks changes, and that the density is fixed as the disk radius changes.

Python was also used to perform sigmoidal and exponential curve fittings starting from the Scipy `curve_fit` function.

Algorithms

Several changes were made to the algorithm over the course of the development of the programme to simulate percolation as accurately as possible.

In order to simulate an infinite system, the function `find_percolation` generates disks with centres inside a square which is bigger than the unit square by one radius on each side. If the centres were generated only in the unit square, the area density on the border of the square would be lower than in the centre because circles with centres inside the unit square could “spill out” of the square but the opposite would not happen. Moreover, a bigger region of space is simulated by diminishing the size of the disks rather than by making the square larger, as it is easiest to work with a unit square.

In order to simulate extensive physical systems, small disk sizes need to be handled. Since iterating over `multiple_runs_fixed_density` with very small disk sizes takes a long time, fewer iterations were performed with small disk sizes than with larger disk sizes so that the total number of disks times the iterations was approximately constant across all data points.

Because the number of disks must be an integer, in order to keep the density constant in `multiple_runs_fixed_density`, the radius interval with which data points were

taken could not be constant. The number of disks had to be rounded to yield an integer and the disk radius had to be recalculated accordingly. When the radius is very large (and the number of disks is consequently low), this could sometimes lead to two successive radius intervals yielding the same effective radius. To avoid there being two y-values for the same radius, the code was programmed in such a way that after one y-value had been generated for a certain radius, other values were skipped until a different number of whole disks could be generated.

Optimisations

To optimise the programme, the source code was instrumented for profiling with the Python `cProfile` function, which showed that more than 90% of the computing time was spent on the function `check_overlap`. This function was optimised, for example by calculating the value of four radius squared outside the loop, and by removing a square root from the distance calculation. After these and other optimisations, computing time was cut down to about one-third of the original time.

In addition to the optimisations made to `check_overlap`, the function `find_percolation` was modified to stop searching for clusters once it finds a percolating cluster unless it is called by `single_run`.

These optimisations enabled a greater number of iterations to be performed in the data analysis code within a reasonable amount of time. This gives the data obtained from these calculations greater statistical accuracy.

Future optimisations

The optimisations mentioned above are coding optimisations. In the future, some algorithm optimisations could be implemented to find the percolation threshold even more efficiently. One possibility could be to divide the unit square into several smaller squares of side length greater than the disk diameter. When calling the function `check_overlap`, only the disks within the same square or within neighbouring squares of the disk being considered should be taken into account.

Results

The percolation threshold for an infinite percolating disk system was calculated in two different ways according to the definition of percolation threshold as the density at which the probability of having long-range connectivity is scale invariant[3].

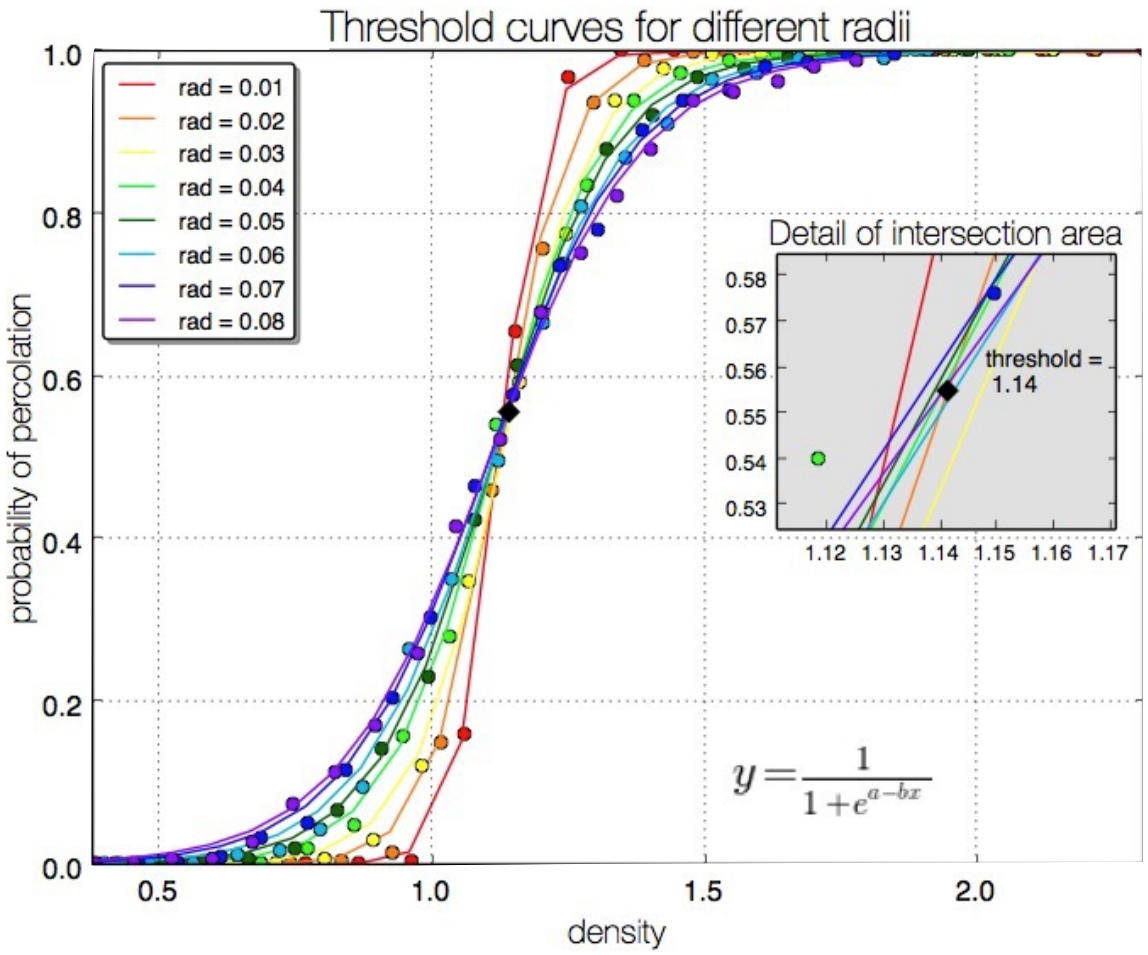
The function `multiple_runs_fixed_radius` was used to generate eight scatter plots of probability of percolation versus disk area density for different disk radii from 0.01 to 0.08 inclusive (see *Graph 1*). Each plot was then fitted to a sigmoid function of the form $y=\frac{1}{1+e^{a-bx}}$.

As the radius of the disks becomes smaller, the sigmoids tend towards a step function[4]. An infinite system represented by a step function would thus have a clear

---

[3] *Project A, Percolation Modelling*. First Year Imperial Python Manual. Available from Blackboard.

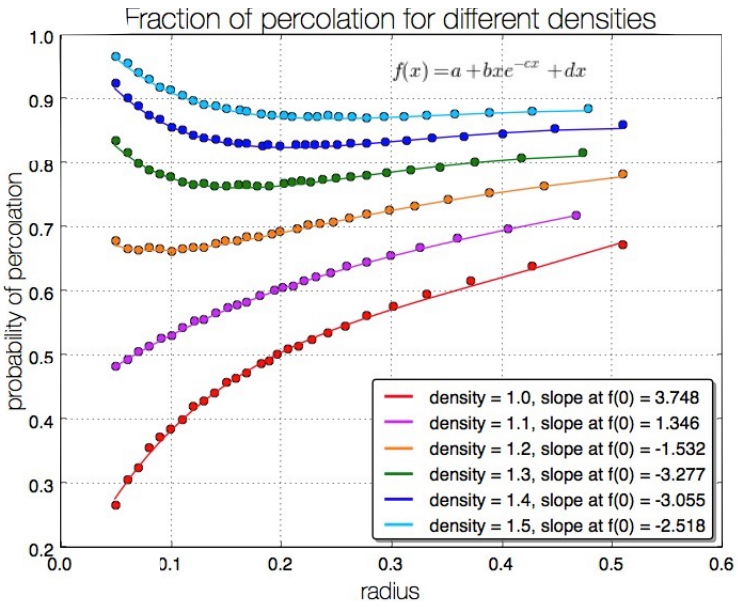
4 *Chapter 2: Percolation*. Available from: http://www.ifb.ethz.ch/education/IntroductionComPhys/2007HS/20071026_script04.pdf



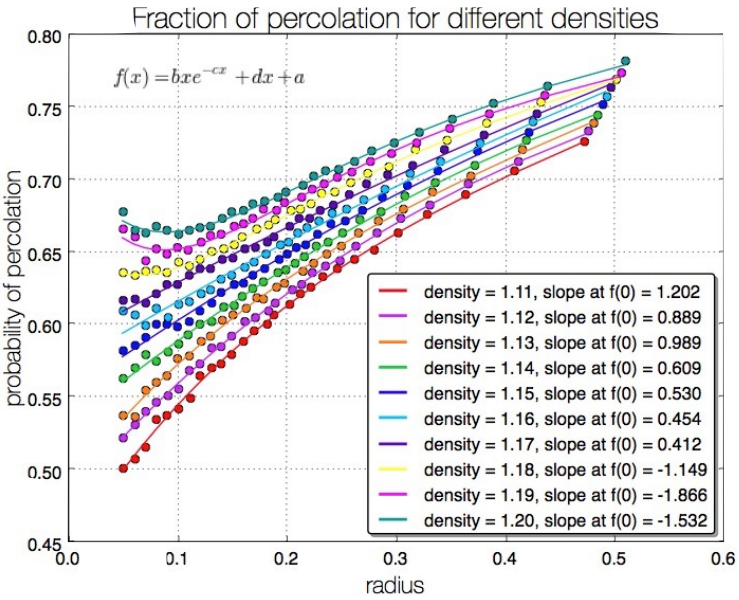
Graph 1: Percolation probability vs. area density. Each curve represents a different disk radius and the percolation probability for each point is calculated from 500 iterations.

threshold density where the step function is discontinuous. Because such a system cannot be simulated on a real computer as it would take an infinite amount of time, *Graph 1* was used to calculate the percolation threshold. Ideally, all the sigmoids should intersect in one point whose x-coordinate is the percolation threshold. With a limited number of iterations, however, the sigmoids intersect in different points, as can be seen in the detail of the intersection area in *Graph 1*. In order to find the threshold, the intersection between each pair of sigmoids was calculated and the points obtained were averaged to yield a value of 1.14 for the percolation threshold.

The function `multiple_runs_fixed_density` was also used to calculate the percolation threshold for an infinite system. In *Graph 2*, five curves show the probability of percolation of systems of fixed density at the variation of the disk radius. These curves were fitted to a function of the form $y=a+bxe^{-cx}+dx$ and their slope was calculated at f(x) = 0 because as the curve approaches zero, the disk radius approaches zero thus simulating an infinite system. Following the definition given for threshold density above, at the threshold density, the probability of percolation for an infinite system is scale invariant. This means that the derivative of the functions calculated must be zero at x = 0. From *Graph 2*, it can be deduced that the slope becomes zero between a density of 1.1 and 1.2. In order to find a more accurate value



Graph 2: Probability of percolation at the variation of the radius. Curves with area densities from 1.0 to 1.5 are shown. The first points are an average of 10,000 iterations and for all successive points, the iterations are inversely proportional to the area of the disks.



Graph 3: Probability of percolation at the variation of the radius. Curves with area densities from 1.11 to 1.20 are shown. The first point is an average of 10,000 iterations and for all successive points, the iterations are inversely proportional to the area of the disks.

for the threshold, ten more fixed-density functions were generated for densities between 1.1 and 1.2. These are plotted in *Graph 3*, where the slope goes to zero between a density of 1.17 and 1.18. Since 1.17 is closer to f(0) = 0 than 1.18, this value was taken as the approximate result for the second method of threshold density computation.

Sources of errors

The values of 1.14 and 1.17 calculated above should not be considered accurate within a high degree of certainty as they are based on statistical simulations with limited numbers of iterations. Both values nevertheless lie within 4% of the currently accepted value of 1.12808737(6) for the threshold for 2D continuum disk percolation calculated by Mertens and Moore[5].

[5] Martens S, Moore C. *Continuum Percolation Thresholds in Two Dimensions*. 2012. Available from: http://arxiv.org/pdf/1209.4936.pdf. (Page 3)

Both values lying within less than 3% of each other despite the fact that they were calculated from two independent sets of randomly generated data, and both lying above the accepted threshold, hints towards a systematic error rather than an error caused by random fluctuations in data points.

The limited number of significant figures of a Python float number cannot account for this error because the values calculated have a lot fewer significant figures.

One source of systematic errors could be that the curves fitted to the data points did not accurately describe the behaviour of the scatter plots. In the simulations with fixed radius, the function to be fitted could be another sigmoidal-shaped function that is not of the form $y=\frac{1}{1+e^{a-bx}}$. Similarly, in the simulations with fixed density, the behaviour of the scatter points might change as the radius approaches zero and a function of the form $y=a+bxe^{-cx}+dx$ may not be an appropriate fit. Future analyses of this problem should calculate the deviation from the various functions (correlation coefficient) in order to find the most proper fit for the data.

Bibliography

(1) Bollobas B, Riordan O. *Percolation*. Cambridge: Cambridge University Press. 2006.

(2) Martens S, Moore C. *Continuum Percolation Thresholds in Two Dimensions*. 2012. Available from: http://arxiv.org/pdf/1209.4936.pdf. [Accessed December 13th, 2015].

(3) *Percolation Theory*. Wolfram alpha. Available from: http://mathworld.wolfram.com/PercolationTheory.html [Accessed January 8th, 2016].

(4) Year 1 Computing Manual. Imperial College London.

(5) The Python Standard Library. Available from: https://docs.python.org/2.7/library/index.html. [Accessed December 2015 - January 2016]