Sam J Shapiro

Spring 2019

Percolation Sample Generation Documentation

1. Introduction:

This document provides an explanation and instructions to the MATLAB code I wrote to generate samples to test percolation in superconducting quantum phase transitions.

2. The Critical p:

The code in question is designed to generate samples that display percolating behavior; where grid sites are randomly occupied with some probability, p, and at some certain critical probability, pc, a connecting path opens across the grid. At this point the behavior of the system usually changes significantly. In this case we expect the sample to go superconducting at this point. Before giving the value of pc, we must decide what it means for two grid sites to be touching. We may choose to only include sites that share an edge (nearest neighbors) as touching. **With only nearest neighbors pc = 0.592**. We may additionally include sites that share a corner but not an edge (next-nearest neighbors). **Including next-nearest neighbors pc = 0.407**. We believe the latter case to be more likely in this experiment, but it is worth considering both.

3. Explanations:

1. “create\_sample.m”:

This code generates a *n × m* array of random 1s and 0s, where the probability of a given entry being 0 is independent from all other entries. This probability is taken as an input. This array can be treated as a black and white image with resolution *n × m* - which may be viewed using MATLAB’s imshow() function. Black pixels represent regions of the sample that contain superconducting material, and white pixels represent regions that do not.

1. “create\_sample\_weighted.m”:

This code is similar to the above, but it allows for pixels to be activated with a higher probability if they are close to other activated pixels. It starts by randomly activating a small amount of pixels. Then, for each remaining non-activated pixel, it calculates the distance to the nearest activated pixel. Pixels are then activated according to a distribution based on distance to the nearest activated pixel. This distribution can be exponential or gaussian, as a function of distance. The resulting images should appear more ‘clustered’ than those generated by create\_sample.m.

1. “iterate\_clusters.m”:

This code is essential copied from part of create\_sample\_weighted.m. It takes an existing image (of the type generated by the above functions), and creates a new image with more activated pixels. These new pixels are activated based on the distance from previously activated pixels, with the same distributions as those in create\_sample\_weighted.m.

4. Instructions:

1. “create\_sample.m”:

This function takes three inputs - m, n, and p. The generated image will be m by n pixels in size. p (a number between 0 and 1) is the probability that any given entry will be activated.

This function returns two outputs - sample and p\_actual. sample is the m by n array that this function is designed to generate. It can be viewed with imshow(sample), and saved with imwrite(sample, ‘filename.png’). p\_actual gives the precise probability that a pixel is activated in sample; this is likely slightly different than the user selected p, as these images are generated randomly.

1. “create\_sample\_weighted.m”:

This function takes six inputs - m, n, p\_i, p\_0, c, and distribution. The generated image will be m by n pixels in size. The code initially activates a small number of random pixels with a probability p\_i. The possible inputs of distribution are “gaussian” and “exponential”, which determines the probability that a pixel will be activated as a function of distance from the nearest initially activated pixel. p\_0, and c are parameters that determine said probability distribution: for “gaussian” and for “exponential”, where is the distance from the given pixel to the nearest activated pixel.

This function returns four outputs - sample, p\_actual, sample\_initial, and comparison. sample is the final m by n array, after the additional pixels have been activated. p\_actual gives the precise probability that a pixel is activated in sample. sample\_initial is an m by n array with only the activated pixels being those that were originally placed with probability p\_i. comparison is another image; it shows sample and sample\_initial side by side, just for an easy visual comparison.

Tips:

* I usually set p\_i = 0.05.
* p\_0 is essentially the probability that a pixel next to an initially activated pixel will be activated. I usually set it between 0.5 and 1.
* c determines how wide the distribution is - the larger c is, the more “clustered” the image will appear. But increasing c will decrease the total number of activated pixels, so it may be good to increase p\_0 as well. I often start with c = 1, and then adjust from there.
* The function defaults to “gaussian” as the distribution. Gaussian is a good first choice, but you can use the regular exponential if less clustering is needed.
* These images can be viewed and saved the same way as those from create\_sample.m.

1. “Iterate\_clusters.m”:

This function takes three inputs - img, p\_0, c, and distribution. img is a previously generated array. p\_0, c, and distribution function the exact same way as in create\_sample\_weighted.m.

This function returns four outputs - new\_img, p\_new, and comparison. new\_img is the array generated by activating more pixels of img. p\_new gives the precise probability that a pixel is activated in new\_img. comparison is another image; it shows new\_img and img side by side, just for an easy visual comparison.