Physics 750: Exercise 17

Tuesday, November 28, 2017

In this exercise, you will explore the phenomenon of *percolation*. The setup is a square lattice of size L from which some fraction of sites has been removed. If that fraction f is large enough, it's possible that the collection of remaining sites is no longer continuously connected. Instead, the sites break up into separate islands of clusters.

Suppose that the size of each cluster is denoted S_α, where the index ranges over N_{cl} clusters. Simple counting requires that

$$L^2(1 - f) = \sum_{\alpha=1}^{N_{cl}} S_\alpha.$$

What we find is that there are two regimes for the cluster sizes: $S_\alpha = O(L^2)$ and $N_{cl} = O(1)$ for small f; $S_\alpha = O(1)$ and $N_{cl} = O(L^2)$ for large f. In the former case, there are a small number of clusters, macroscopic in size, and there will typically be a *spanning* cluster that touches two opposing edges of the square lattice. In the latter case, there are a macroscopic number of small clusters, none of which reach across the lattice. In the bulk limit ($L \to \infty$), these two regimes are separated by a sharp transition at a critical fraction $f_c \approx 0.407$.

Rather than work with a fixed removal fraction f, it is simpler to consider a probability of removal p (in part, because f can't be varied continuously on a finite lattice). A new disorder configuration can be created by sweeping through each site of the lattice, drawing a random number $\xi \leftarrow [0, 1]$, and removing the site if $\xi < p$. The quantities of interest are the size of the largest cluster and the cluster size moments, averaged over many disorder realizations:

$$\bar{S}_{max} = \left\langle \max_\alpha S_\alpha \right\rangle, \quad M_2 = \left\langle \sum_{\alpha=1}^{N_{cl}} S_\alpha^2 \right\rangle, \quad M_4 = \left\langle \sum_{\alpha=1}^{N_{cl}} S_\alpha^4 \right\rangle.$$

Here, $\langle \cdot \rangle$ denotes the disorder average. You'll find that these quantities exhibit very different scaling for $p < p_c \equiv f_c$ and for $p > p_c$. Most amazing, the dominant cluster at $p = p_c$ is a fractal with dimension $d_f = 91/48$.

Identifying the clusters is equivalent to assigning an index $I \in \{1, 2, \cdots, N_{cl}\}$ to each of the sites such that $I_i = I_j$ iff the sites i and j are in the same cluster. Of course, i and j are in the same cluster iff there exists a chain of intermediate nearest-neighbour sites connecting the two. This leads rather nicely to a recursive algorithm based on the observation that every site is considered to be in the same cluster as its four nearest neighbours.

- define $A(i, I)$ as the following procedure:

    - if i is a valid site in the lattice and $I_i = 0$ then assign $I_i := I$
    - call $A(i + \hat{x}, I)$, $A(i - \hat{x}, I)$, $A(i + \hat{y}, I)$, and $A(i - \hat{y}, I)$

- loop over all sites i from 1 to L^2

    - assign $I_i := 0$

- assign $I := 1$

- loop over all sites i from 1 to L^2

    - if i is a valid site in the lattice and $I_i = 0$ then call $A(i, I)$
    - assign $I := I + 1$

What's recursive about this? It's the fact that the procedure A calls itself. A simpler example should make this clear: the factorial of n is defined by $0! = 1! = 1$ and $n! = 1 \cdot 2 \cdots n$. This mathematical operation can be implemented as a traditional function.

```
unsigned long int factorial(unsigned long int n)
{
   unsigned long int fact = 1;
   for (unsigned long int m = 2; m <= n; ++m)
      fact *= m;
   return m;
}
```

Since $n! = n \cdot (n-1)!$, it can also be implemented as a recursive function.

```
unsigned long int factorial(unsigned long int n)
{
   if (n < 2) return 1;
   return n*factorial(n-1);
}
```

1. Use the `curl` command to download from the class website everything you'll need for this exercise.

   ```
   $ WEBPATH=http://www.phy.olemiss.edu/~kbeach/
   $ curl $WEBPATH/courses/fall2017/phys750/src/exercise17.tgz -O
   $ tar xzf exercise17.tgz
   $ cd exercise17
   ```

   The `make` command will create three executables `perc1`, `perc2`, and `perc3`. The first displays an animation showing the evolution of the cluster structure on a 50×50 lattice as sites are removed one at a time. Each connected cluster is given its own colour. The second program shows a still image of the cluster structure on a 100×100 lattice for one particular disorder realization in which sites have been removed with probability p. The third program is only partially written. It compiles but doesn't do anything yet.

2. Run `./perc1` and watch for the disintegration of the spanning cluster (as $f : 0 \to 1$) into many isolated islands. Using `perc2`, observe the cluster structure for particular choices of p. Try the following values.

   ```
   $ ./perc2 0.32
   [ctrl-c]
   $ ./perc2 0.40
   [ctrl-c]
   $ ./perc2 0.48
   [ctrl-c]
   ```

3. In `perc1.cpp`, add code to the function `build_clusters` that determines whether there is a spanning cluster and sets the boolean variable `is_spanning` accordingly. One strategy is to compare the `cluster_membership` values of all points on the left and right edges and on the top and bottom edges of the lattice to see if there is a value in common.

   The function `display` uses `is_spanning` to control its text output. When you've successfully completed the modification, you should see something like

   removal fraction = 0.285 clusters = 31 (spanning)

   removal fraction = 0.617 clusters = 278

4. Modify `perc3.cpp` so that progam also calculates the *size* of each cluster. You can do this any way you wish, but the following approach is particularly simple. You'll notice that clusters are constructed recursively using a function `assign_membership` that calls itself. Change the prototype to read

```
//void assign_membership(size_t i, size_t j, long int m)
long int assign_membership(size_t i, size_t j, long int m)
```

and have the function return 1 plus the number of sites connected to each of its nearest neighbours.

5. The file perc3.cpp contains no graphics code. It's supposed to output the values $\bar{S}_{max}$, M_2, M_4 averaged over 100 disorder realizations for particular values of L and p. Make the necessary changes to build_clusters so the program functions as intended. When you're done, you should get output like the following.

```
$ ./perc3
Usage: perc3 (0 < L < 500) (0 <= p < 1)
$ ./perc3 12 0.25
         103.84            10943.4            1.23475e+08
$ ./perc3 96 0.66
          33.19            20944.9            5.44033e+06
```

6. The shell script transition.bash outputs the quantities L, p, $\bar{S}_{max}$, M_2, M_4 in 5-column format. The data are arranged in blocks of increasing L that sweep over the values $0 \leq p < 1$. For example, to plot $\bar{S}_{max}/L^2$ as a function of p, we would execute the following commands.

```
$ ./transition.bash > tr.dat
Computing L = 10
Computing L = 20
Computing L = 30
Computing L = 60
Computing L = 100
Computing L = 200
Computing L = 300
$ gnuplot
gnuplot> plot "tr.dat" using 2:3
gnuplot> plot "tr.dat" using 2:($3/$1**2) with lines
```

Try plotting M_2/L^2 and M_4/L^4 versus p as well. Use the Binder ratio $U = M_4/(M_2)^2$ to pinpoint the critical value of p.

7. The shell script scaling.bash outputs the same quantities but now organized in blocks of constant p and sweeping over L. What do the following plots signify?

```
$ ./scaling.bash > sc.dat
Computing p = 0.1
Computing p = 0.2
Computing p = 0.3
Computing p = 0.35
Computing p = 0.40725379
Computing p = 0.45
Computing p = 0.5
Computing p = 0.6
Computing p = 0.7
$ gnuplot
gnuplot> plot "sc.dat" using (1.0/$1):($3/$1**2) with lines
```

```
gnuplot> set logscale
gnuplot> plot "sc.dat" using 1:3 with linespoints, \
         0.45*x**(91/48.), 0.9*x**2
```