PHYS580 Lab10 Report

Yicheng Feng
PUID: 0030193826

October 27, 2019

Workflow: I use the Linux system, and code C++ in terminals. For visualization, I use the C++ package – ROOT (made by CERN) to make plots. At the end, the reports are written in LaTeX.

The codes for this lab are written as the following files:

- `percolation_2d.h` and `percolation_2d.cxx` for the class `Percolation2D` to simulation the percolations and calculate $P(p)$ and $S(p)$.
- `lab10.cxx` for the main function to make plots.

To each problem of this lab report, I will attach the relevant parts of the code. If you want to check the validation of my code, you need to download the whole code from the link `https://github.com/YichengFeng/phys580/tree/master/lab10`.

(1) First, set the site occupation probability to $p = 0.593$ (very close the percolation threshold p_c for a 2D lattice), and study percolation on lattices of several different sizes (from $L \times L = 50 \times 50$ through 1000×1000 or so). Generate at least 20 realizations for each lattice size, and for each percolation record the percolation probability $P(p_c)$ and the susceptibility $S(p_c)$ (the latter is essentially the mean cluster size). Then, average P and S over the set of realizations, and determine how the averaged $P(p_c)$ and $S(p_c)$ depend on the lattice edge size L for asymptotically large L. This dependence is called finite-size scaling (you should find power-law behavior). Estimate the power-law exponents by linear least-squares fitting on log-log scale.

Theoretically, the expected asymptotic behaviors are

$$P(p_c) \sim L^{-\beta/\nu}, \quad S(p_c) \sim L^{\gamma/\nu} \tag{1}$$

Physics explanation:

I wrote the program with C++ from scratch with the following steps

- Generate all occupied sites with p. (time complexity $\mathcal{O}(N)$, $N = L^2$)
- Use depth first search (DFS) to find all clusters. ($\mathcal{O}(N)$)
- Find the largest cluster, and calculate the P and S.

The scan of L is did from $L = 50$ to $L = 1000$. Figure 1 shows two representitive results of the percolation with the largest cluster in orange.

Figure 2 shows the L dependence of the percolation probability with $p = p_c = 0.593$. The right pad is the log-log plot with linear fitting, which gives the slope $-\beta/\nu = -0.10664$. The theoretical value is $\beta = 5/36$, $\nu = 4/3$, and $-\beta/\nu = -5/48 \approx -0.104167$. Therefore the simulation result is very close to the exact result.

Figure 3 shows the L dependence of the percolation probability with $p = p_c = 0.593$. The right pad is the log-log plot with linear fitting, which gives the slope $\gamma/\nu = 1.79156$. The theoretical value is $\gamma = 43/18$, $\nu = 4/3$, and $\gamma/\nu = 43/24 \approx 1.79167$. Therefore the simulation result is very close to the exact result.

Plots:

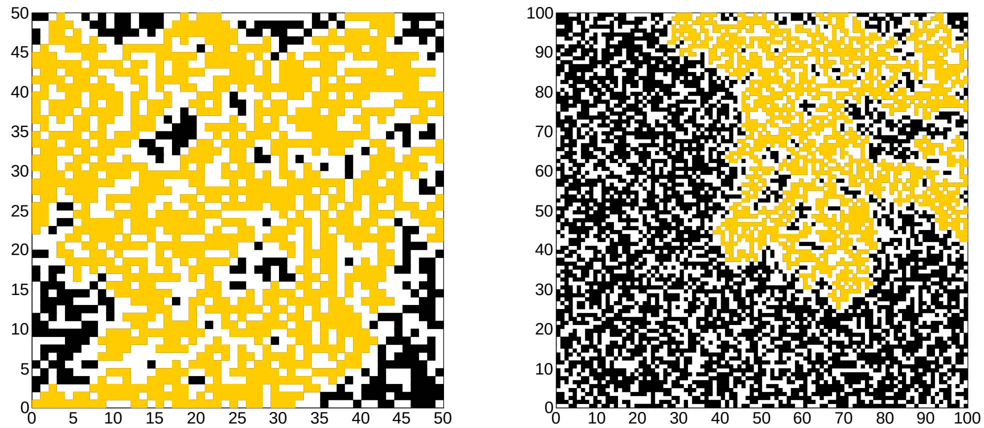

Figure 1: Some representative results of the percolation sites: left pad 50×50, right pad 100×100. The orange means the largest cluster with $p = p_c \approx 0.593$.

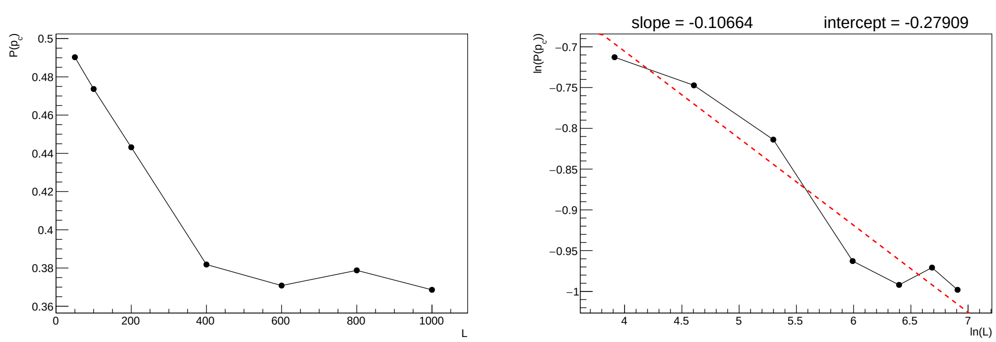

Figure 2: The percolation probability $P(p_c)$ depends on the lattice edge L with $p = p_c \approx 0.593$.

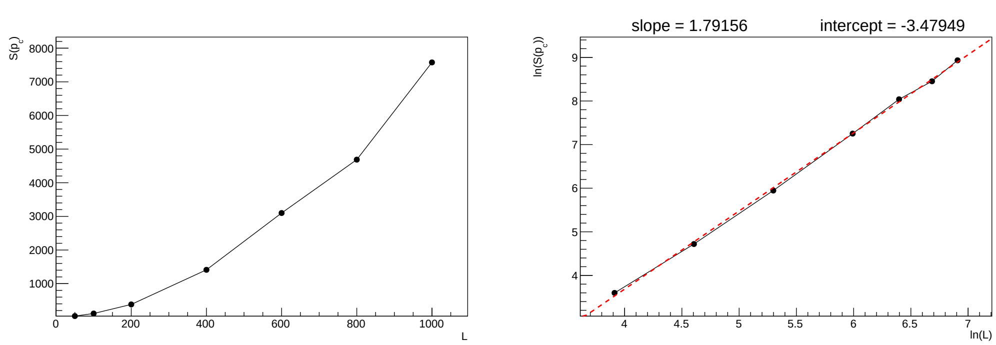

Figure 3: The percolation susceptibility $S(p_c)$ depends on the lattice edge L with $p = p_c \approx 0.593$.

Relevant code:

For the generation of the occupied sites

```
//----------------------------------------------------------------------//

void Percolation2D::cal_perculation() {

        for(int i=0; i<_L; i++) {
                for(int j=0; j<_L; j++) {
                        if(1.0*rand()/RAND_MAX<_p) {
                                _occupied[i][j] = true;
                        } else {
                                _occupied[i][j] = false;
                        }
                }
        }
}

//----------------------------------------------------------------------//
```

For the DFS to find all clusters

```
//----------------------------------------------------------------------//

void Percolation2D::cal_DFS(int ix, int iy, int cid) {

        if(ix<0 || ix>=_L) return;
        if(iy<0 || iy>=_L) return;

        if(_cluster_id[iy][ix] == 0 && _occupied[iy][ix]) {
                _cluster_id[iy][ix] = cid;
        } else {
                return;
        }

        cal_DFS(ix+1, iy, cid);
        cal_DFS(ix-1, iy, cid);
        cal_DFS(ix, iy+1, cid);
        cal_DFS(ix, iy-1, cid);
}

//----------------------------------------------------------------------//

void Percolation2D::cal_cluster() {

        for(int i=0; i<_L; i++) {
                for(int j=0; j<_L; j++) {
                        _cluster_id[i][j] = 0;
                }
        }

        int cid = 0;
        for(int iy=0; iy<_L; iy++) {
                for(int ix=0; ix<_L; ix++) {
                        if(_cluster_id[iy][ix]==0 && _occupied[iy][ix]) {
                                cid ++;
                                cal_DFS(ix, iy, cid);
                        }
                }
        }

```

```
        _cluster_size.clear();
        _cluster_x.clear();
        _cluster_y.clear();
        _spanning_cluster_id.clear();
        for(int i=0; i<=cid; i++) {
                _cluster_size.push_back(0);
                _cluster_x.push_back(vector<int>());
                _cluster_y.push_back(vector<int>());
        }
        for(int iy=0; iy<_L; iy++) {
                for(int ix=0; ix<_L; ix++) {
                        int idx = _cluster_id[iy][ix];
                        _cluster_size[idx] ++;
                        _cluster_x[idx].push_back(ix);
                        _cluster_y[idx].push_back(iy);
                }
        }

        int max = 0;
        int max_id = 1;
        for(int i=1; i<=cid; i++) {
                if(_cluster_size[i]>max) {
                        max = _cluster_size[i];
                        max_id = i;
                }
        }
        for(int i=1; i<=cid; i++) {
                if(_cluster_size[i]==max) {
                        _spanning_cluster_id.push_back(i);
                }
        }
        _n_cluster = cid;
}

//------------------------------------------------------------------------//
```

For calculation of the P and S

```
//------------------------------------------------------------------------//

void Percolation2D::cal_PS() {

        if(!check()) return;

        _P = 1.0*_cluster_size[_spanning_cluster_id[0]]/_n_occupied;

        double sum2 = 0;
        double tmpi = 0;
        for(int i=1; i<_cluster_size.size(); i++) {
                if(tmpi < _spanning_cluster_id.size() && i == _spanning_cluster_id[tmpi]){
                        tmpi ++;
                        continue;
                }
                sum2 += 1.0*_cluster_size[i]*_cluster_size[i];
        }
        _S = sum2/_L/_L;
}

//------------------------------------------------------------------------//
```

(2) Next, generate percolation realizations for at least 10 different values of p around p_c on large lattices of fixed given size $L \times L$. Plot $P(p)$ and $S(p)$ (averaged over at least 20 realizations at each p) as a function of p, and show that for sufficiently large L these two quantities behave as power laws near (but not too near!) $p = p_c$. Specifically, $P(p) \sim (p - p_c)^\beta$ for $p > p_c$, and $S(p) \sim |p - p_c|^\gamma$ on both sides of p_c . From your numerical data, estimate the critical exponents β and γ. Combine the results with your findings from part (1) to estimate ν as well.

Physics explanation:

In this problem, we use lattice with $L = 100$. Each case is repeated 100 times, and the percolation probability $P(p)$ and susceptibility $S(p)$ are averaged over those trials. The occupation probability p scan is did in two ranges $p > p_c$ and $p < p_c$, with 10 samples. The lower range is $0.24 \leq p \leq 0.493$, and the higher range is $0.596 \leq p \leq 0.83$.

Figure 4 shows the percolation probability $P(p)$ depends on occupation probability p. The right pad shows the log-log plot and the linear fitting. The slope is 0.18717. The theoretical value for the slope is $\beta = 5/36 \approx 0.13889$. The difference seems reasonable. From the simulation results of percolation probability P, we can estimate the value of ν:

$$-\beta_1/\nu_1 = -0.10664, \quad \beta_1 = 0.18717, \quad \Rightarrow \quad \nu_1 = 0.18717/0.10664 = 1.75516, \tag{2}$$

where the exact value should be $\nu = 4/3 \approx 1.33333$, relative error is 32%.

Figure 5 shows the percolation susceptibility $S(p)$ depends on p with normal scale, while Fig. 6 shows the log-log plot with linear fitting. In the low p range, the linear fitting is good, and the slope is -2.39785 close to the theoretical value $-\gamma = -43/18 \approx -2.3889$. However, in the high p range, the linear fitting is bad and the slope is -2.30303 a little away from the exact value. We can use the good fitting of low p range to estimate ν

$$\gamma_2/\nu_2 = 1.79156, \quad \gamma_2 = 2.39785, \quad \Rightarrow \quad \nu_2 = 2.39785/1.79156 \approx 1.33841, \tag{3}$$

where the exact value should be $\nu = 4/3 \approx 1.33333$, relative error is 0.4%.

Plots:

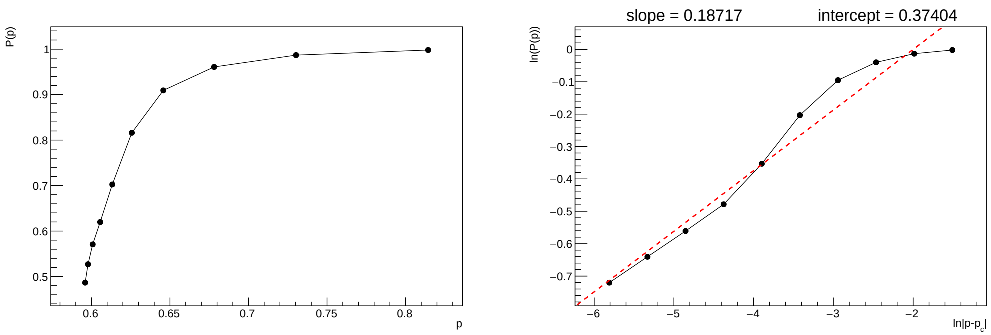

Figure 4: The percolation probability $P(p)$ depends on the occupation probability p with $L = 100$ and $p > p_c$. The right pad is log-log plot.

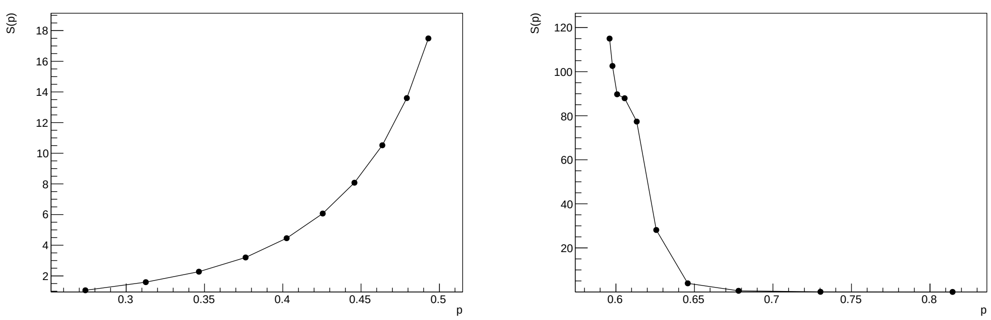

Figure 5: The percolation susceptibility $S(p)$ depends on the occupation probability p with $L = 100$. Left pad: $p < p_c$; Right pad: $p > p_c$.

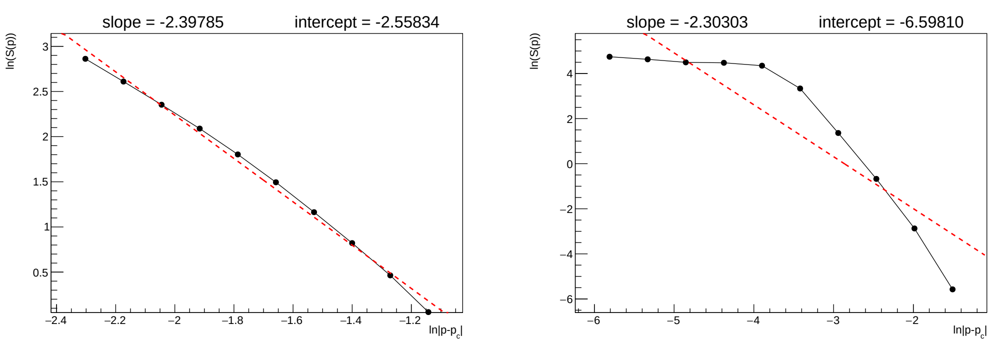

Figure 6: The percolation susceptibility $\ln S(p)$ depends on the occupation probability $\ln |p - p_c|$ with $L = 100$. Left pad: $p < p_c$; Right pad: $p > p_c$.

Relevant code:

The code of the percolation simulation has been shown in problem (1).

For the initialization with various p

```
//------------------------------------------------------------------------//

Percolation2D::Percolation2D(double p, int L, int trial) {

        _p = p;
        _L = L;
        _trial = trial;

        for(int i=0; i<_L; i++) {
                vector<bool> tmp_occ;
                vector<int>  tmp_cid;
                for(int j=0; j<_L; j++) {
                        tmp_occ.push_back(false);
                        tmp_cid.push_back(0);
                }
                _occupied.push_back(tmp_occ);
                _cluster_id.push_back(tmp_cid);
        }
}

//------------------------------------------------------------------------//
```