1 Percolation

We only want to create a matrix where each point is on or off with the probability of P. The code is simple and only consists of a for loop and a heatmap. In class we discussed that the percolation algorithm that first comes to mind is highly inefficient and we don't need to code it.

1.1 Results

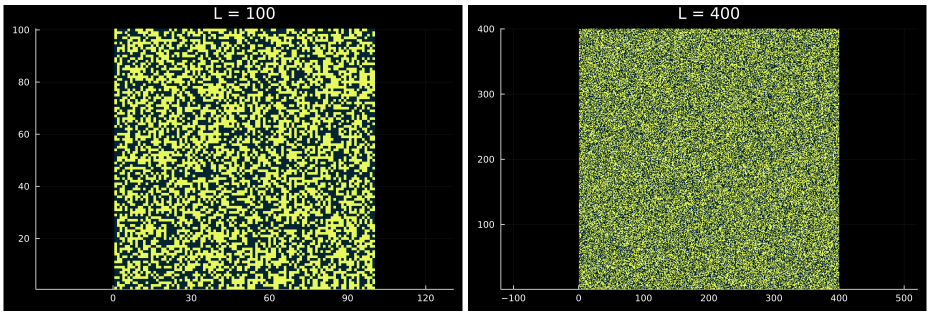

Figure 1: Results for randomly filled grid.

2 Percolation Coloring

The first clustering algorithm we use is the coloring algorithm. First we generate a randomly filled matrix using the code used in the previous part. Then we use the coloring algorithm to check if percolation has happened.

2.1 neighbourcheck() function

This function takes a matrix and the coordinates of a cell to check if the neighbouring cells of that cell are non zero. Then it returns an array containing the values of the non zero neighbouring cells and the minimum amount in this array.

2.2 neighbourmin() function

This functiontakes a matrix and the nonzero array as an input, then it sets the number of all the cells corresponding to the values in the nonzero array and sets them all to the minimum amount in said array. It then returns the resulting matrix.

2.3 Coloring() function

This function takes the initial matrix as input and permorms the coloring algorithm on it using the neighbourcheck() and neighbourmin() functions. It then returns the resulting matrix.

2.4 Plotting

We then fill the cells with the value 1 with a large numb er to make them more apparent. We also check if the final column of the matrix contains the value 1 to evaluate if percolation happened. We then plot the resulting matrix using a heatmap.

2.5 Results

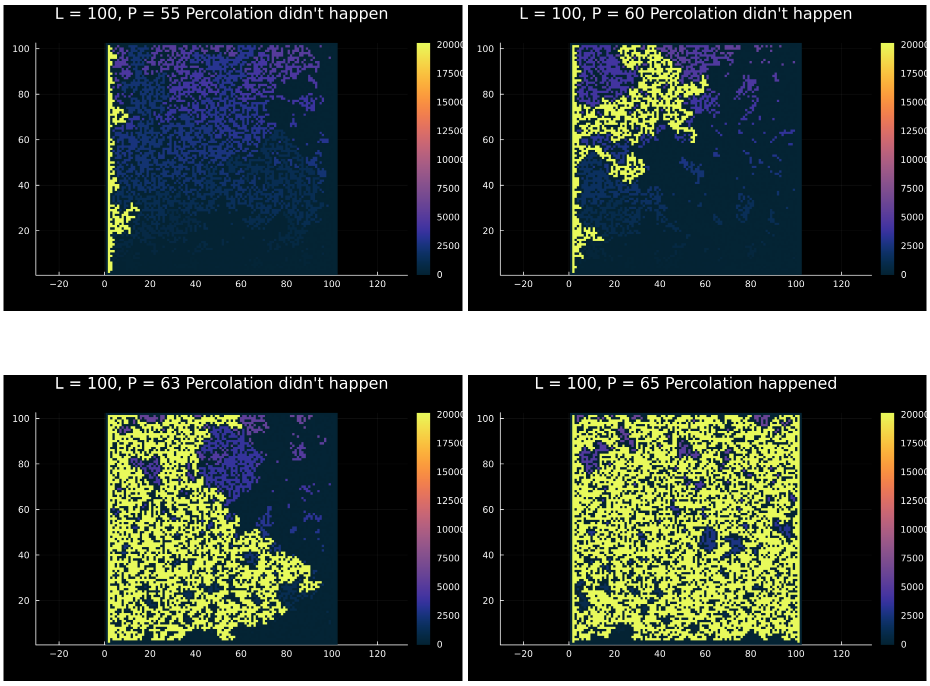

Figure 2: Results for percolation using the coloring algorithm.

3 Hoshen-Kopelman and Q

In order to calculate Q effectively, we must first implement the Hoshen-Kopelman algorithm. We won't go in depth of the inner workings of the algorithm, but we must use a union-find algorithm to label the clusters and merge them. This algorithm is implemented in the Hoshen-Kopelman.jl file and the results are as follows:

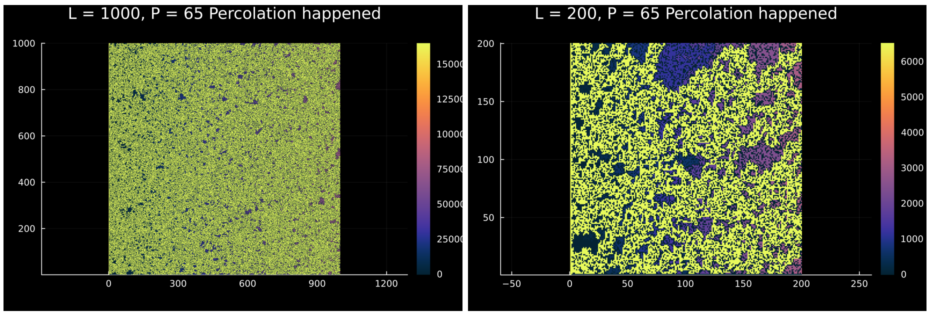

Figure 3: Results for the Hoshen-Kopelman algorithm.

3.1 Q.jl file

To calculate and plot the values of Q we use the Q.jl file that uses the Hoshen-Kopelman algorithm.

3.2 perc() function

This function takes L and P as input, then constructs a randomly filled matrix and uses the Hoshen-Kopelman algorithm to check for percolation. Then it returns 1 if percolation happened or 0 if it didn't happen.

3.3 Averaging and Plotting

We use a for loop to run the algorithm for 40 values of P from zero to 100, 100 times each. Then the data is averaged and plotted using a scatter plot for different amounts of L.

3.4 Results

The bottom label is incorrectly scaled from 0 to 2 but it's actually in the range of 0 to 1.

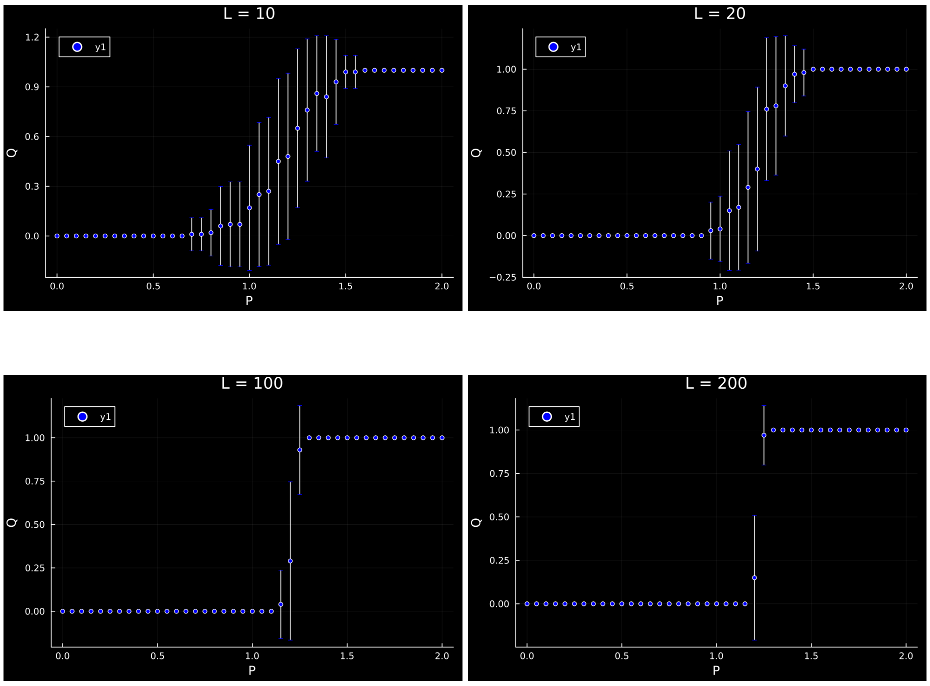

Figure 4: Results for Q based on P.

4 Qinf.jl

This file is used to calculate and plot the values of Q_∞. The code is really similar to the code used in the previous part so we wouldn't go in depth. The only difference is that in this code, the perc() function also returns the label matrix. There is also a function called count() that counts the number of cells with value 1 in the matrix.

4.1 Results

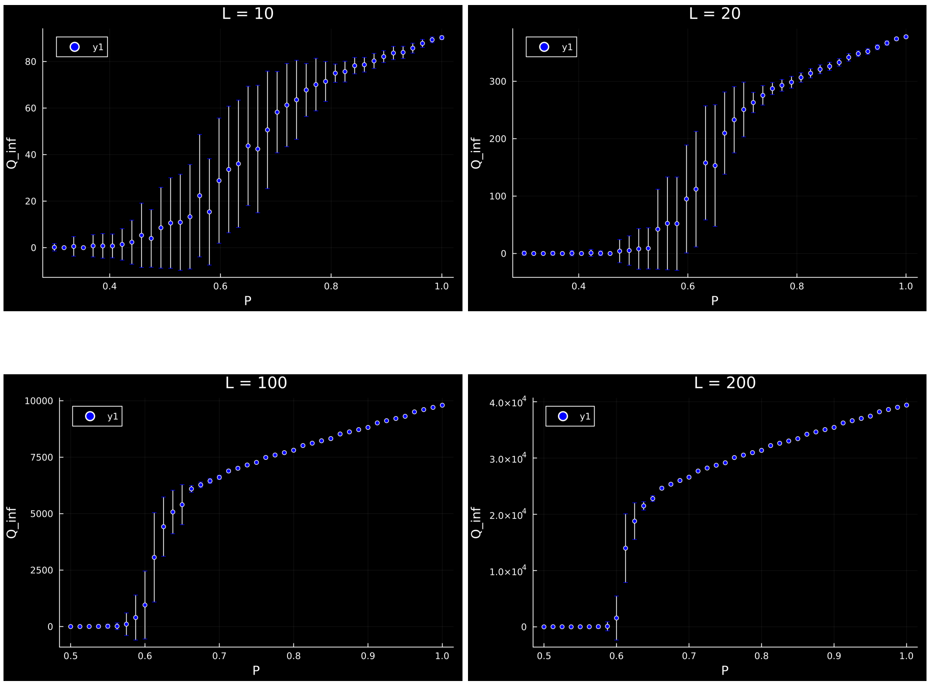

Figure 5: Results for Q_∞ based on P.

5 Correlation Length and Critical Point

In this part we also use the Hoshen-Kopelman algorithm and the same averaging loop we have used up until now but we also define a function to find the radius of gyration of the clusters then average over them.

5.1 correlationlength() function

This function takes a matrix and the number of the label of the cluster we want to analyze, then it finds the coordinates of all of the cells in that cluster and finds the radius of gyration of the cluster around its mean. Then it returns the radius.

5.2 exists() function

This function takes the matrix and a label as an input, then searches the matrix for that label. if the label exists it returns 1, if there isn't it returns 0.

5.3 Results

This code has a bug that returns NaN for some zs and I ran out of time when debugging so the results are not optimal and will not be shown. If the results existed, we could have derived ν by finding the peaks and plotting them based on $|P - P_c$ on a log log scale and fitting a line.

6 Cluster Growth

6.1 creatematrix() function

This function creates the initial matrix needed to start the growth which is an $L + 2 \times L + 2$ matrix with a dot at the middle and walls around it.

6.2 grow() function

This function takes the coordinates of a point in the created matrix and executes the growing algorithm recursively until the function runs out of available growing options. This code uses the value 0 as an empty cell, 1 as an occupied cell and 2 as a blocked cell. The growth is as follows:

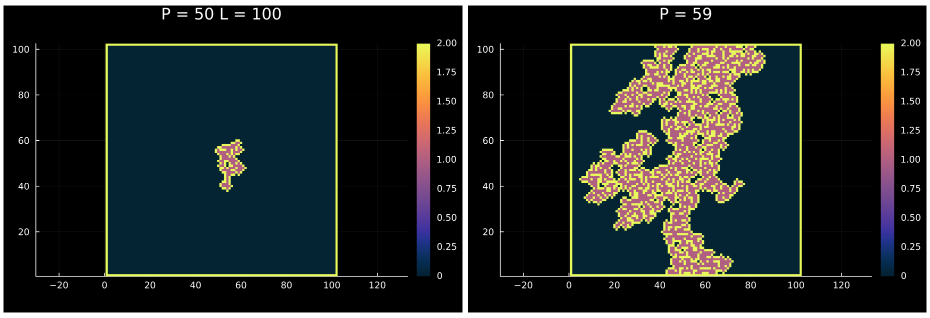

Figure 6: examples of growth.

6.3 correlationlength() function

This is the same as the function defined in the previous part with the difference that this only looks for clusters filled with 1.

6.4 findsize() function

This function takes the matrix as input and returns the size of the cluster.

6.5 Plotting

We run the function 1000 times and delete the data that results in NaN. then we plot the data on a scatter plot and fit a line to the curve to find the dimension.

6.6 Results

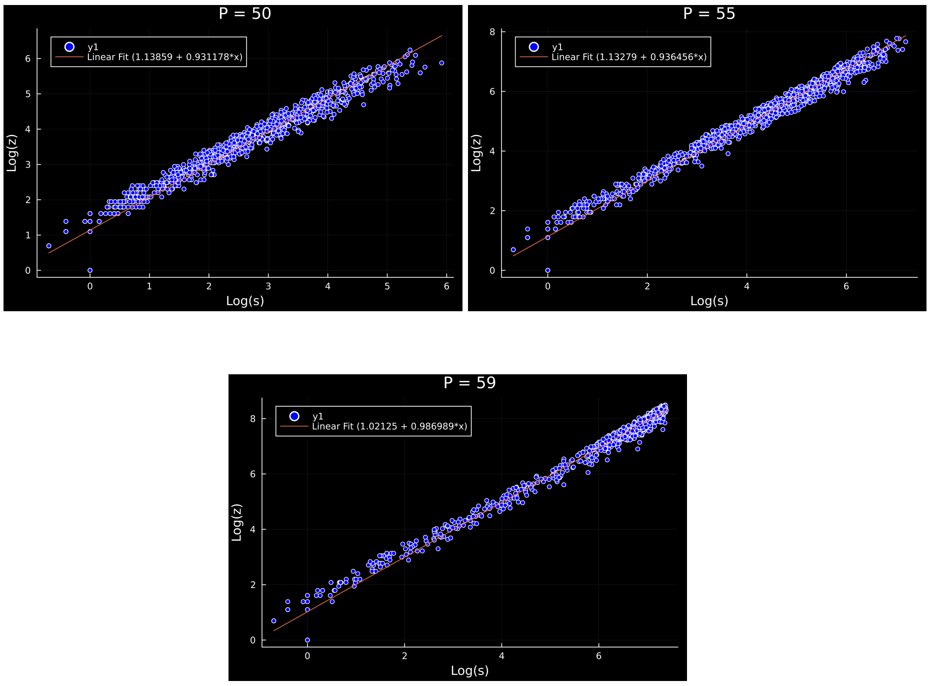

Figure 7: Results for the growth algorithm.