AP157 Output 1 - January 30, 2026

Benedict A. Pangilinan (bapangilinan5@up.edu.ph)

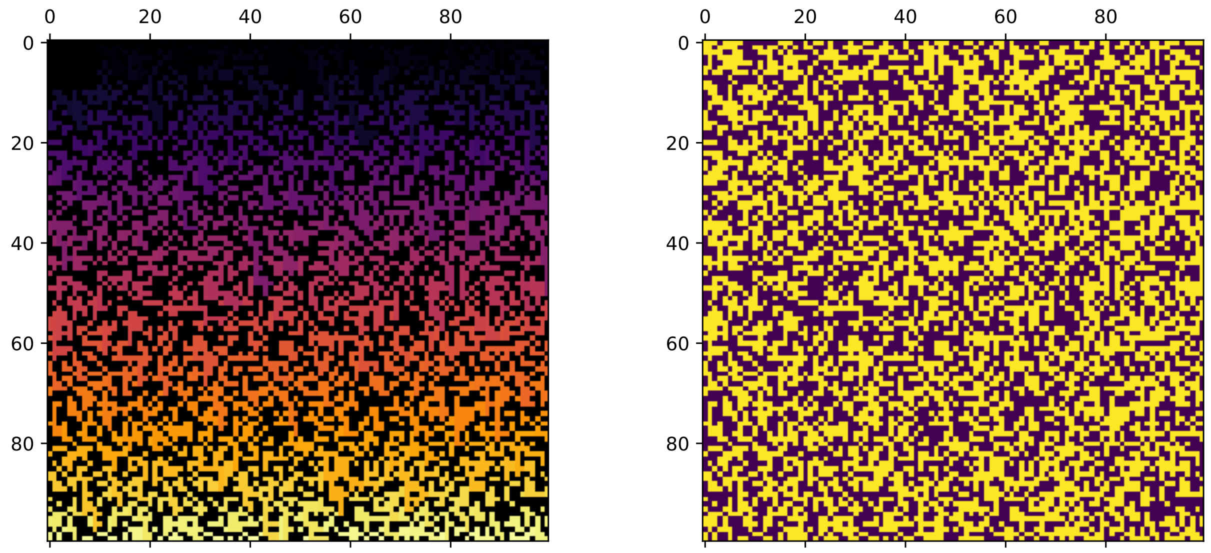

Figure 1: **Percolation behavior** In the left, we can see the occupied pixels in a 100×100 frame. Meanwhile on the right, is the corresponding percolation matrix.

Clustering algorithm

The hitting of the percolating clusters from top to bottom is shown as follows:

```
#Checking for a cluster that spans top to bottom
# Determine clusters present at the top row
round1 = []
for i in range(len(top_row)):
  if top_row[i] != 0:
      count = 0
      for entry in round1:
        if top_row[i] == entry:
          count = 1
      if count == 0: # append qualifiers if they're
           not a repetition of the existing
        round1.append(int(top_row[i]))

#Do they also exist on the bottommost row?
landers = []
for entry in round1:
  for key in bottom_row:
    #If they do, then it spans!
    if entry == key:
      count = 0
      for strike in landers: # for a round1 entry
          that got into bottom row
        if entry == strike:
          count = 1 # means no longer unique
      if count == 0:
        landers.append(entry)
```

They're repeated for the leftmost column and rightmost column due to the them being non-periodic.

Probability Increase

Not shown here, but by plotting the number of clusters vs. the occupation probability, we observe that the frequency of incidence peaks at between 17 to 25 hits at around p of 0.4. Considering, we had percolation matrices of $N = 10$, iterated across thousands of probabilities. The plot has shown an extremal behavior, as expected from large sample size, albeit a bit skewed to the left. The code used is as follows:

```
# Generate percolation matrices at different
    occupation probabilities
probability_space = np.arange(0,1,0.001) # We'll
    scatter 1000 pts.
results = []
for occupation_probability in probability_space:
    results.append(clustering(10,
        occupation_probability))
```