Check the connection

- **Topic:** percolation, phase transitions, critical points
- **Task:**
    1. Create and analyse some systems with different node occupation probabilities p to get familiar with the simulation. Estimate roughly, at which p percolation happens.
    2. Implement a function to run a series of simulations at different p.
    3. Calculate estimates for the percolation probability P and its statistical error as a function of node probability p.
    4. Based on the results, estimate the critical probability p_c.
- **Template:** percolation.py
- **Further reading:**
    - https://en.wikipedia.org/wiki/Percolation
    - https://en.wikipedia.org/wiki/Percolation_theory
    - https://en.wikipedia.org/wiki/Percolation_threshold
    - https://en.wikipedia.org/wiki/Critical_phenomena

percolation.py

`percolation.find_top_cluster(grid)` [source]

Find and mark the lattices sites with an unbroken connection to the top of the system.

Two occupied sites (value 1) are connected, if they are directly adjacent, i.e., connected by a common side left-right or up-down.

Two sites are connected by an unbroken connection, if there is and unbroken chain of connected sites between the two.

This algorithm starts from the top of the system and systematically searches for all the sites with an unbroken connection to the top.

All the connected sites will be given the value of 2 for later recognition.

Parameters:: **grid** (*array*) – the system as an integer lattice

`percolation.generate_network(lattice_size, node_probability)` [source]

Creates a random $N \times N$ square lattice.

Each lattice site is given a random value of 0 or 1. A value of 0 represents an unoccupied site. A value of 1 represents an occupied site.

Parameters::
- **lattice_size** (*int*) – system length N
- **node_probability** (*float*) – probability for each node to get value 1

Returns:: the network
Return type:: array

`percolation.is_vertically_connected(grid)` [source]

Checks if there is an unbroken connection between the top and bottom sides of the system.

The function first calls the function `find_top_cluster()`. to locate the percolation cluster. Then, this routine checks if this cluster reaches the bottom of the lattice.

Parameters:: **grid** (*array*) – the system as an integer lattice
Returns:: True if a connection exists
Return type:: bool

`percolation.percolation_mean_and_error(successes, repeats)` [source]

Calculates the average and error of mean for percolation probability.

Assume that the percolation probability for the system is P. If you generate n systems, on average, you will find percolation nP times. The precise number is, however, a random variable.

Let us define p as the percolation result in a *single simulation*. If there is percolation, $p = 1$, and if not, $p = 0$. Let us also denote the total number of systems with $p = 1$ as $k = \sum_p p$. The average of p is then

$$\mu_p = \frac{1}{n}\sum_p p = \frac{k}{n}.$$

Since on average $k = nP$, we have $P = k/n$, and so the average of p is an unbiased estimate for the true probability P.

Similarly, the sample variance of p is, by definition,

$$\begin{aligned} s_p^2 &= \frac{1}{n-1}\sum_p (p-\mu_p)^2 \\ &= \frac{1}{n-1}\sum_p \left(p^2 - 2p\frac{k}{n} + \frac{k^2}{n^2}\right) \\ &= \frac{1}{n-1}\left(k - 2\frac{k^2}{n} + \frac{k^2}{n}\right) \\ &= \frac{k}{n-1}\left(1 - \frac{k}{n}\right) \\ &= \frac{k(n-k)}{n(n-1)}. \end{aligned}$$

(Note $\sum_p p^2 = \sum_p p = k$, and $\sum_p 1 = n$.)

The standard error of mean for our estimated μ_p is then

$$\Delta p = \frac{s_p}{\sqrt{n}} = \frac{1}{n}\sqrt{\frac{k(n-k)}{(n-1)}}.$$

Parameters::
- **successes** (*int*) – number of systems with percolation, k
- **repeats** (*int*) – total number of systems, n

Returns:: sample mean μ_p, error of mean Δp
Return type:: float, float

`percolation.print_progress(step, total)` [source]

Prints a progress bar.

Parameters::
- **step** (*int*) – progress counter
- **total** (*int*) – counter at completion

`percolation.run_series(p_min, p_max, p_steps, lattice_size, n_repeats)` [source]

Runs a series of simulations and calculates the probability that a connection between the top and bottom of the system exists.

The method varies node occupation probability p and creates, for each probability, M independent systems. These systems are $N \times N$ square lattices. For each system, the algorithm checks if a connection exists. Based on the results, the function estimates the percolation probability P and its error estimate using `percolation_mean_and_error()`.

Finally, the function plots the calculated percolation probabilities P as a function of node probability p.

Note

This function is incomplete!

Parameters::
- **p_min** (*float*) – smallest node probability p to check
- **p_max** (*float*) – highest node probability p to check
- **p_steps** (*int*) – number of different probabilities to check
- **lattice_size** (*int*) – system size N
- **n_repeats** (*int*) – number of random systems to create for each probability, M

Returns:: percolation probabilities P, error estimates ΔP
Return type:: list, list

`percolation.run_simulation(node_probability, lattice_size)` [source]

Creates and analyses a single system.

The created system will be a $N \times N$ square lattice where each site is occupied with probability p.

The function reports if there is a connection between the top and bottom of the system and shows a picture of the system.

Parameters::
- **node_probability** (*float*) – node occupation probability p
- **lattice_size** (*int*) – system size N

Returns:: the created lattice
Return type:: array

`percolation.show_network(grid)` [source]

Plots the system.

Parameters:: **grid** (*array*) – the system as an integer lattice

Quick search
Go