Exercise 2: Swendsen-Wang algorithm for the 2D Ising model

The goal of this exercise is to implement the Swendsen-Wang algorithm for the 2D Ising model. Recall that the update of the Swendsen-Wang algorithm works as follows:

- Assign to each bond b a variable $w_b \in \{0, 1\}$, where 1 means "connected" and 0 means "disconnected". If the spins connected by the bond $b = (n, m)$ between index n and m are anti-parallel, always choose $w_b = 0$, If the spins are parallel, choose $w_b = 0$ with probability $e^{-2\beta J}$, where $\beta = \frac{1}{k_B T}$. In terms of conditional probabilities:

$$\begin{aligned} p(w_b = 0|\sigma_n \neq \sigma_m) &= 1, & p(w_b = 0|\sigma_n = \sigma_m) &= e^{-2\beta J}, \\ p(w_b = 1|\sigma_n \neq \sigma_m) &= 0, & p(w_b = 1|\sigma_n = \sigma_m) &= 1. - e^{-2\beta J}. \end{aligned}$$

- Interpreting the connected bonds as edges of a graph (where 0 means no edge) and the spins as node, find "clusters" of spins, i.e., connected components of the graph.
- Flip each cluster (= connected component) with probability $p = 0.5$. Notice that a single spin is also a "cluster".

After the update, do a measurement (if the system is already thermalized) and continue with the next update.

a) Setup the system by writing these three functions

   - `def xy_2_idx(int x, int y)` $\rightarrow$ `int idx`
   - `def idx_2_xy(int idx)` $\rightarrow$ `int x, int y`
   - `def create_bond_indices(int Lx, int Ly)` $\rightarrow$ `list bond_indices`

   Since we need to identify clusters, it is better to label the lattice sites (x, y) by a single integer number $n = n(x, y) = x \cdot L_y + y = 0, 1, \cdots, N - 1$ where $N = L_x \cdot L_y$. The inverse mapping is given by $x(n) = n//L_y$ (integer division, rounding down) and $y(n) = n \mod L_y$. Create an array that contains the indices of each bond (n, m) of the square lattice with periodic boundary conditions exactly once. There are in totals 2N bonds and we indicate each bond by two indices. The array for bond indices is just for keeping track of the indices of the bond and is immutable. It should not be confused with the bond configuration, i.e. each bond connected or not. The array for bond indices should have shape $(2N, 2)$, e.g.

$$\overbrace{\begin{pmatrix} 0, & 1 \\ 1, & 2 \\ \vdots & \vdots \\ L_y - 1 & 0 \\ \vdots & \vdots \end{pmatrix}}^{\text{size=2}}$$

b) Initialize a 1D array for the spin states σ_n (having random entries ± 1).

c) Write a function that determines, i.e. sample, the bond configuration w_b for each of the bonds, given the spin configuration.
`def update_bond_config(np.ndarray spin_config, np.ndarray bond_indices, float T)` $\rightarrow$ `np.ndarray bond_config`

d) To determine the connected components, you can use the function `scipy.sparse.csgraph.connected_components`. Look up the documentation of the function online. The graph is represented by a sparse matrix of the type `scipy.sparse.csr_matrix`. You can use the initialization of the form `csr_matrix((bond_config, (bond_indices[:, 0], bond_indices[:, 1])), shape=(N, N))`. Don't forget to add the transposed to obtain a symmetric graph. Using the result of `connected_components`, flip each of the determined clusters with probability $p = 0.5$. Collect the code in a function performing one update with the Swendsen-Wang algorithm.

e) Write functions to measure the energy and magnetization.

f) Write a function to run the whole Monte Carlo simulation at given temperature, returning arrays with all measured values E and M. Plot the mean values for different temperatures and make sure you get (roughly) the same results as last week. (Begin the comparison with small systems!)

g) Optionally: If in previous functions, you construct the function with for loop and in else clauses, now is a good time to optimize the code with `numba.jit`.

h) Measure and plot the auto-correlation of the energy

$$C_E(\delta) = \frac{\langle E_{t+\delta} E_t \rangle_t - \langle E_t \rangle_t^2}{\langle (E_t)^2 \rangle_t - \langle E_t \rangle_t^2}$$

versus δ (and similarly for the magnetization) for T right at, above, and below the critical temperature T_c.

i) Recreate the the auto-correlation plots for the update with the Metropolis algorithm. Notice that to compare the global update algorithm, e.g. Swendsen Wang algorithm, with the local update algorithm, e.g. Metropolis algorithm, the result of Metropolis algorithm should be measured with $N = L_x \cdot L_y$ updates.