Problem Set 4

Submit your solution via `NTULearn`, with one Python source file per problem. Problems labeled like this—(5* marks)—are optional for undergraduates, and can be attempted for a bonus of 1/5 the indicated marks; for graduate students, the problems are compulsory and receive full marks. Follow good programming style, and label output plots clearly.

0. PAGERANK

PageRank is an algorithm for deciding the "importance" of web-pages. It treats importance as a recursive concept: a page is *important* if many other *important* pages linking to it.

Suppose we have N pages, labeled $n = 0, 1, 2, \cdots, N-1$. Each page links to other pages. Let $\mathcal{I}_n$ be the pages linking to page n ("inbound" links to n), and $\mathcal{O}_n$ the pages that page n links to ("outbound" links from n). The PageRank for page n, denoted by $\mathcal{P}_n$, is defined as

$$\mathcal{P}_n = \frac{1-d}{N} + d \sum_{m \,\in\, \mathcal{I}(n)} \frac{\mathcal{P}_m}{|\mathcal{O}_m|}, \tag{0}$$

where $d \in [0,1]$ is an adjustable "damping" parameter and $|\mathcal{O}_m|$ denotes the size of $\mathcal{O}_m$. Note that $\mathcal{P}$ appears on both the left- and right-hand sides of this equation.

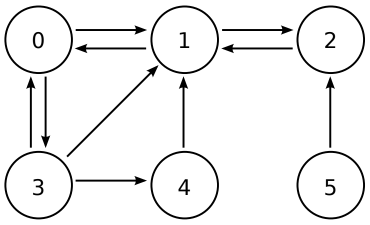
According to Eq. (0), each page boosts the PageRank of the pages that it links to. This boost is divided equally among the out-going links; e.g., if you link to 100 pages, the PageRank boost you give to each individual linked page is diluted by a factor of 100. The boost is also weighted by the PageRank of the linking page; thus, your page gets a big boost if the *New York Times* site links to you, but a tiny boost if my personal blog links to you.

We can compute the PageRank with a Markov process. Let each page represent a Markov state. Start from a random page; on each step, where we are on page n, do the following:

1. With probability $1-d$, randomly jump to any one of the N pages (with equal weight).
2. Otherwise (with probability d), pick one of the out-going links $m \in \mathcal{O}_n$ (with equal weight), and jump to page m.

Upon writing down the detailed balance equation for this Markov process, we find that the steady-state probability for state n is exactly the PageRank $\mathcal{P}_n$ defined in Eq. (0). Hence, we can compute PageRank by running the Markov chain and tracking the proportion of time spent on each page.

(a) (4 marks) Write a function `pagerank_demo(d=0.1)`, which computes PageRank on the specific $N = 6$ network shown on the right; numbered circles indicate pages, and arrows indicate links. The input `d` specifies the PageRank algorithm's damping parameter d. For each page $n \in \{0, 1, 2, 3, 4, 5\}$,



- Plot the value of $\mathcal{P}_n(T)$ generated by the Markov chain, versus the number of Markov chain steps T. Choose an appropriate range for T.
- In each plot, show also (as a horizontal line) the exact PageRank computed using the transition matrix.

(b) (6 marks) We will now run the PageRank algorithm on a sample of $\sim 10^4$ Wikipedia articles. The file `wikilinks.pickle` contains information about a set of Wikipedia articles and the links between them. It can be loaded with the following Python code:

```
import pickle
with open("wikilinks.pickle", "rb") as f:
    titles, links = pickle.load(f)
```

The variable `titles` is a list of Wikipedia article titles, such that `titles[n]` is a string containing the title of article n. The variable `links` is a list, such that `links[n]` is a list of integers specifying the articles that article n links to. For example, if `links[50]` has the value `[3, 7, 99]`, that means that article 50 links to articles 3, 7, and 99 (and you can retrieve the titles of those articles at `titles[3]`, `titles[7]`, and `titles[99]`).

Write a function `pagerank_wikipedia_demo(d=0.1)`, which finds and prints the titles of the "most important" 200 Wikipedia articles in the sample, along with their PageRanks. The input `d` specifies the damping parameter d.

(c) (*3 marks) The file `physicists.pickle` stores a list of physicist names, in no particular order. Modify `pagerank_wikipedia_demo` so that it additionally prints the names of the 20 "most important" physicists, along with their PageRank scores.

1. 2D ISING MODEL

The 2D Ising model consists of N spins arranged in a square lattice. Let S_i denote the spin at site i, which can have the value +1 or -1. The system state is specified by the values of all the spins, $\{ S_i \}$. Assuming no external magnetic field, the total energy is

$$E = -J \sum_{\langle ij \rangle} S_i S_j, \tag{1}$$

where $\langle ij \rangle$ denotes pairs of nearest-neighbour i and j sites (without double-counting pairs). Apart from the energy, we're also interested in the total magnetization

$$M = \frac{1}{N} \sum_i S_i. \tag{2}$$

We will assume the lattice has periodic boundary conditions: e.g., if the lattice is $N_x \times N_y$, then site $(0, 0)$ is neighbours with the sites $(1, 0)$, $(N_x - 1, 0)$, $(0, 1)$, and $(0, N_y - 1)$.

(a) (10 marks) Write a function

```
def ising_mc(J=1., Nx=16, Ny=16, nsteps=50000):
```

Inputs	
`J`	Value of the J Ising model parameter.
`Nx, Ny`	The values of N_x and N_y, i.e. the number of sites in the x and y directions respectively.
`nsteps`	The total number of steps per Monte Carlo simulation.

The function should perform Monte Carlo simulations of the 2D Ising model at various temperatures T. You may hard-code an appropriate range for T (a good choice is between $1/J$ and $3.5/J$); the Boltzmann constant shall be normalized to $k_B = 1$. The function should then produce the following plots:

- $\langle M \rangle$ versus T.
- The heat capacity C_B versus T. The heat capacity is defined as $C_B = \left[\langle E^2 \rangle - \langle E \rangle^2 \right] / T$.
- The magnetic susceptibility $\chi_m = \langle M^2 \rangle - \langle M \rangle^2$.

Simulations should be performed using the Metropolis algorithm. When calculating averages, you may wish to discard some "early" steps to let the Markov chain settle towards the stationary distribution.

Furthermore, uring each Monte Carlo step, be sure not to re-calculate the total energy E for the whole lattice. That is not necessary; you only need the energy difference resulting from flipping spin i, which can be determined using the values of the neighbouring spins.

Discuss in code comments: How do the various thermodynamic quantities behave with temperature?

(b) (7* marks) "Re-weighting" is a method for using a Monte Carlo trajectory, performed with temperature T_0, to get information about the system's behavior at a different temperature T_1. Suppose $A(E)$ is some function of the total energy; using properties of the partition function, one can show that the mean value of A at temperature T_1 is

$$\Big\langle A(E)\Big\rangle_{T_1} \approx \frac{\Big\langle A(E)\, e^{-(\beta_1-\beta_0)E}\Big\rangle_{T_0}}{\Big\langle e^{-(\beta_1-\beta_0)E}\Big\rangle_{T_0}}, \tag{3}$$

where $\beta_n \equiv 1/T_n$ and $\langle\dots\rangle_{T_0}$ denotes an average taken at temperature T_0. The two $\langle\dots\rangle_{T_0}$ quantities on the right-hand side of Eq. (3) can be computed during a Monte Carlo run at temperature T_0. Thus, we can estimate $\langle A(E)\rangle_{T_1}$ without having to do a separate Monte Carlo run at temperature T_1. In fact, a single Monte Carlo run can provide the data for many different temperatures at once.

Modify your code from part (a) so that the plot of C_B versus T also includes the estimated heat capacity from re-weighting. The re-weighting estimates should be obtained by a single Monte Carlo run (a good choice is $T_0 \sim 2.4/J$, but you can experiment with other values). Label the various curves clearly.