CS166 Problem Set 2

Instructions

- Show all your writing, math, code, and results in a Python/Jupyter notebook. Upload the .ipynb (or .zip) file of the notebook.
- Your instructor has to be able to run your code. Do not simply upload a PDF version of your Python notebook.
- Complete all 4 problems below. You will be graded on the LOs and HCs specified in each problem. You may footnote additional HCs if you wish.

Problem 1: Rainfall records

- LOs: #Modeling, #EmpiricalAnalysis
- HCs: #distributions
- Words: 150–300; and code

We explore how record rainfalls (highest ever or lowest ever) are distributed.

1. Do a bit of research online (cite your sources) and come up with a plausible continuous probability distribution for the annual rainfall (mm per year) in Berlin. Motivate your choice of distribution.

2. Next, simulate the annual rainfall in Berlin for 101 consecutive years (labeled Year 0 up to Year 100) by generating i.i.d. samples from your chosen probability distribution.

   A rainfall value is a record high if it is greater than those in all previous years (starting with Year 0), and a record low if it is lower than those in all previous years. In the century starting from Year 1 and ending with Year 100 (including those two years), produce a histogram to show the distribution over the number of "record" years – i.e., the number of years that have <u>either</u> a record high <u>or</u> a record low.

   Show empirically that the expected value of this distribution is

   $$2 \sum_{i=1}^{100} \frac{1}{i+1} \approx 8.395$$

   (You do not have to prove this mathematically. Show that your simulation results confirm the value above.)

3. Next, use your rainfall distribution to simulate the first year after Year 0 when a record high (not low) rainfall occurs. Produce a histogram of the distribution over the number of years until the first record high after Year 0.

   Show empirically, using a line plot on top of your histogram, that this distribution has probability mass function

   $$P(k) = \frac{1}{k\,(k+1)}$$

4. Explain why your results in Questions 2 and 3 do <u>not</u> depend on your choice of rainfall distribution in Question 1.

Problem 2: Metropolis-Hastings

- LOs: #PythonImplementation, #CodeReadability
- Words: 100–200; and code

Your task is to generate samples from a 2-dimensional probability distribution using the Metropolis-Hastings sampling algorithm.

- Implement the Metropolis-Hastings algorithm.
- Explain what your proposal distribution is and why you chose it.
- Explain the output from the algorithm, including how long it takes to reach equilibrium and how many steps you take between samples (and why).

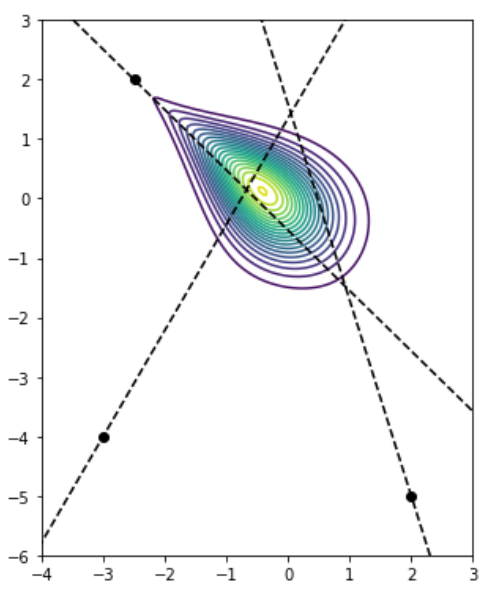
The normalization constant of the distribution is not known. Use the Python function below to compute the unnormalized distribution.

```
def unnormalized_distribution(x, y):
    from numpy import arctan2, pi
    from scipy.stats import norm
    measurements = [2.349080, -1.276010, -2.081865]
    buoys = [[-2.5, 2], [2, -5], [-3, -4]]
    sigma = 10/180*pi
    posterior = 1
    for i in range(3):
        bearing = arctan2(buoys[i][1] - y, buoys[i][0] - x)
        difference = (bearing - measurements[i] + pi) % (2 * pi) - pi
        posterior *= norm.pdf(difference, loc=0, scale=sigma)
    return posterior
```

For context, this distribution arises from the following Bayesian inference problem.

You are on a boat and measure your bearings (directions) to 3 buoys. Your measurements are not very accurate and the error in the measured direction in degrees is Normal(0, 10^2) – so a standard deviation of 10 degrees. Use the known locations of the buoys and your 3 noisy measurements to determine where you are on the sea.

The figure to the right shows the locations of the 3 buoys, the noisy directions measured from the location of the boat to those buoys, and the posterior distribution over the boat's location (calculated by the Python function above) from which you need to generate samples.



Diameter and average distance

- LOs: #TheoreticalAnalysis
- Words: 150–300

When we think about a single aggregate measure to summarize the distance between all the nodes in a graph, there are two natural quantities that come to mind. One is the *diameter*, defined as the maximum (shortest) distance between any pair of nodes in the graph. Another is the *average distance*, which is the average (shortest) distance between all pairs of nodes in the graph.

In many graphs, these two quantities are close to each other in value – but there are exceptions.

1. Provide an example of a graph with at least 6 nodes where the diameter is equal to the average distance. Explain why the metrics are equal.
2. Describe an example of a graph with at least 6 nodes where the diameter is more than 3 times as large as the average distance. Explain why the diameter is more than 3 times the average distance.
3. Describe how you can construct a graph in which the diameter is more than c times as large as the average distance – for any positive value of c. Briefly explain why or prove that the diameter is more than c times the average distance.

Hopfield networks

- LOs: #PythonImplementation, #CodeReadability
- Words: 100–200; and code

A *Hopfield network* is a model that can recover memorized binary state patterns from noisy initial conditions. If a pattern of nodes was memorized and you initialize the network with a similar pattern of nodes (but has some of the node values flipped from –1 to 1 or *vice versa*), the network will recover the memorized pattern.

This was one of the early advances in neural network research and the principles of Hopfield networks – described below – are still used today in various computational neural models.

Here are the typical assumptions made in the Hopfield network model:

- Nodes take on one of two values – either –1 or +1.
- The network is fully connected – every node is connected by an edge to every other node (except itself).
- The edges have weights and the weights are <u>not</u> updated while the simulation runs.
- Node states are updated synchronously in discrete time steps according to this rule:

  $$s_i(t+1) = sign\left(\sum_{j \neq i} w_{ij}\, s_j(t) \right)$$

- Here, $s_i(t)$ is the state of node i at time t, $w_{ij} = w_{ji}$ is the symmetric edge weight between nodes i and j, and $sign(x)$ is a function that gives +1 if $x > 0$ and –1 otherwise.

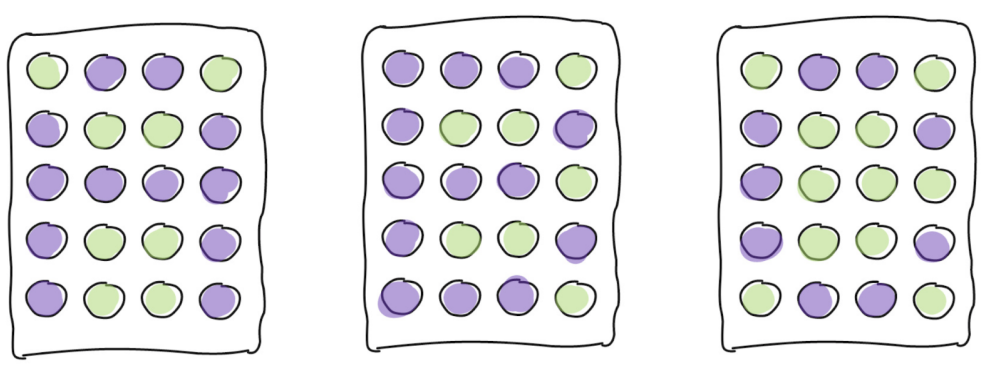
John Hopfield showed that you can store a finite number of "memories" (or patterns) in this network by carefully setting the edge weights using the following formula:

$$w_{ij} = \frac{1}{K} \sum_{k=1}^{K} s_{ik} s_{jk}$$

Here, s_{ik} is the state of node i in the kth pattern (of K total patterns) to be stored in the network. So if two nodes in a pattern are equal (both +1 or both –1), their edge weight is increased but if they are unequal, their edge weight is decreased.

Based on Hopfield's work, the capacity of a network, defined as the number of distinct memories that the network can store without getting errors, is approximately $0.138N$ where N is the number of nodes in the network. Your task is to test this with a small network and a small number of patterns.

For a network with 20 nodes, $0.138N = 2.76$ which means we should be able to store 2 patterns but not 3. Test this hypothesis using the patterns below (the first 3 letters of the Latin alphabet) and show that you can store any 2 of these patterns but not all 3.



To run your tests, start the simulation from a random initial state and show that the network ends up in one of the three patterns above after a few steps – but only if you store any 2 of these 3 patterns and not all 3.

If you run into any problems or interesting behavior in this simulation, be sure to describe your observations and explain why you think they occur.