Example sheet 3a

Stochastic processes
Foundations of Data Science—DJW—2017/2018

- You are **not** meant to answer every question—but you are expected to spend around 4 solid hours on this sheet, including the practical class. You should spend around 2 hours on example sheet 3b.
- There is an accompanying Python notebook `ex3a.ipynb` with useful code snippets. You can find the link on the course webpage. You are not required to program in Python; you should answer the questions first in pseudocode; then, if you have time, implement them in any programming language you like.
- In a Jupyter Notebooks, you can run your code cells in any order. For the sanity of your supervisor, before saving your work, please use Kernel | Restart & Clear Output and re-run your code in the order it appears in the notebook. This will uncover bugs that arise from assigning a variable in one place and using it earlier in the notebook.

For supervisors: This is half an example sheet. It may be supervised after 10 November. The second half may be supervised after the final lecture, 24 November. Model answers can be found on the course webpage.

In the first 6 questions you will make inferences about the location of a moving object, given noisy observations of its location. To keep the maths and the code simple, we will start with the following very simple model:

- Let X_n be the location at timestep n, $X_n \in \{0,1,\dots,9\}$, and let X_0 be uniformly distributed in $\{0,1,\dots,9\}$
- Given $X_n = x$, let $X_{n+1} = \text{next\_state}(x)$ where

  ```
  def next_state(x):
      d = random.choice([−1,1])
      return max(0, min(9, x+d))
  ```

  The max and min ensure that X_{n+1} stays in $\{0,1,\dots,9\}$. The random.choice() function returns one of the values provided, both of them equally likely.
- Let Y_n be the observation at timestep n, $Y_n = \text{noisy\_obs}(X_n)$ where

  ```
  def noisy_obs(x):
      e = random.choice([−1,0,1])
      return max(0, min(9, x+e))
  ```

This is called a *hidden Markov model*. The underlying process X is a Markov chain, but we don't observe it directly, we only get noisy observations, and the noise in each timestep is independent.

Question 1. Let $\delta_0(x) = \mathbb{P}(X_0 = x) = 1/10$, and let $\pi_0(x) = \mathbb{P}(X_0 = x \,|\, Y_0 = y_0)$. (This depends on y_0, so π_0 is really a function of x and y_0, but for the sake of brevity we won't write out this dependency.) Use Bayes' rule to find a formula for π_0 in terms of δ_0 and the matrix $Q_{xy} = \mathbb{P}(Y_n = y \,|\, X_n = x)$.

Question 2. Let $\delta_n(x) = \mathbb{P}(X_n = x \,|\, Y_0 = y_0,\dots,Y_{n-1} = y_{n-1})$. Use the law of total probability to find a formula for δ_1 in terms of π_0 and the transition matrix $P_{xz} = \mathbb{P}(X_{n+1} = z \,|\, X_n = x)$.

Question 3. Let $\pi_n = \mathbb{P}(X_n = x \,|\, Y_0 = y_0,\dots,Y_n = y_n)$. Find a formula for π_n in terms of δ_n and Q.

Question 4. Write pseudocode for a function that accepts δ_0 and a list of readings $y = [y_0,\dots,y_{n-1}]$ and produces a list $[\pi_0,\dots,\pi_{n-1}]$. Your pseudocode should include defining P and Q. *You can see a Python version of this code, and a routine to plot the output, in ex3a.ipynb. To answer this question, you may either produce your own psuedocode from scratch, or you may annotate the Python code with comments explaining what it does.*

Question 5. If your code is given the input $y = [3,3,4,9]$, it should fail with a divide-by-zero error. Give an interpretation of this failure.

It is undesirable for production code to fail with divide-by-zero errors. One way to fix the problem is to modify the Markov model to include a ‘random teleport’—to express the idea ‘OK, our inference has gone wrong somewhere; let’s allow our location estimate to reset itself’. We can achieve this mathematically with the following model: with probability $1-\varepsilon$ let $X_{n+1} = \mathsf{next\_state}(X_n)$, and otherwise let X_{n+1} be chosen uniformly from $\{0,1,\dots,9\}$. Modify your code to reflect this new model, with $\varepsilon = 0.01$.

Alternatively, we could fix the problem by changing the model to express ‘OK, this reading is glitchy; let’s allow the code to discard an impossible reading’. How might you change the Markov model to achieve this?

Question 6. The Markov model for motion that we are using is called a *random walk (with boundaries)*; it chooses a direction of travel independently at every timestep. This is not a good model for human movement, since people tend to head in the same direction for a while before changing direction.

(a) Let $V_n \in \{-1,0,1\}$ be a Markov chain: let $V_{n+1} = V_n$ with probability 0.9, and let V_{n+1} be chosen uniformly at random from $\{-1,0,1\}$ with probability 0.1. Draw a state space diagram for this Markov chain.

(b) Interpret V_n as the velocity of our moving object at timestep n, and let $X_{n+1} = \max(0,\min(9,X_n+V_n))$. Update your code to reflect this model.

Question 7 (stationary distribution). A two-arm bandit is a gambling machine with two levers. It costs 10p to pull a lever, and each lever pays out 20p on a win. The chance of a win is α for lever A and β for lever B, and we don’t know α or β, but we do know they are both in the range $0 < \alpha,\beta < 1$. Here is a gambling strategy: if my last pull resulted in a win then I’ll stick with the lever I just pulled, otherwise I’ll switch.

Draw a state space diagram with two states, for the Markov chain where $X_n \in \{A,B\}$ is the nth lever I pulled. Calculate its stationary distribution.

Question 8 (stationary distribution, limit theorems). In the setting of Question 7, let $Y_n = (X_n, X_{n+1})$. Explain why Y_n is a Markov chain, and draw its state space diagram, and calculate its stationary distribution.

Using the Markov chain Y, or otherwise, find the long-run fraction of lever pulls that result in a payout. Explain your reasoning carefully.

Question 9 (stationary distribution). Consider a directed acyclic graph representing the web, with one vertex per webpage, and an edge $v \to w$ if page v links to page w. Consider a random web surfer who goes from page to page according to the algorithm

```
d = 0.85
def next_page(v):
    neighbours = list of pages w such that $v \to w$
    # random.random() generates a Uniform[0,1] random variable
    if len(neighbours) > 0 and random.random() ≤ d:
        return random.choice(neighbours)
    else:
        V = list of all web pages
        return random.choice(V)
```

[ATTENTION: the exercise sheet as handed out had the wrong condition on line 5.] This defines a Markov chain. Explain why the chain is irreducible and aperiodic. Show that the stationary distribution π solves

$$\pi_v = \frac{1-d}{|V|} + d \sum_{u:u\to v} \frac{\pi_u}{|\Gamma_u|} \tag{1}$$

where $|V|$ is the total number of web pages in the graph, and $|\Gamma_u|$ is the number of outgoing edges from u.

Compute the stationary distribution for this random web surfer model, for the graph in lecture notes Example 4.4. Repeat with $d = 0.05$. What do you expect as $d \to 0$? What do you expect if $d = 1$?

Equation (1) *defines PageRank, Google’s original method for ranking websites. Google said that in 2013 it was indexing more than 30 trillion webpages.*

Question 10 (limit theorems). In Example 4.1 in lecture notes, Markov’s model for text generation, we picked an arbitrary starting string `c=['o','n']`. Suppose instead that we want a random starting string drawn from the stationary distribution, but that the state space is too large for us to be able to compute the stationary distribution explicitly. How can we generate a suitable random starting string?