4. Stochastic processes

Be able to formulate models for stochastic processes, and understand how they can be used for estimation and prediction. Be familiar with calculation techniques for memoryless stochastic processes. Understand the classification of discrete state space Markov chains, and be able to calculate the stationary distribution, and recognise limit theorems.

Science is often concerned with the laws that describe how a system changes over time, such as Newton's laws of motion. When we use probabilistic laws to describe how the system changes, the system is called a *stochastic process*. We used a stochastic process model in Section 1.2 to analyse Bitcoin; the probabilistic part of our model was the randomness of who generates the next Bitcoin block, be it the attacker or the rest of the peer-to-peer network. In Part II, you will come across stochastic process models in several courses:

- in *Computer Systems Modelling* they are used to describe discrete event simulations of communications networks
- in *Machine Learning and Bayesian Inference* they are used for computing posterior distributions
- in *Information Theory* they are used to describe noisy communications channels, and also the data streams sent over such channels.

4.1. Markov chains

Example 4.1. The Russian mathematician Andrei Markov (1856–1922) invented a new type of probabilistic model, now given his name, and his first application was to model Pushkin's poem *Eugeny Onegin*. He suggested the following method for generating a stream of text $C = (C_0, C_1, C_2, \dots)$ where each C_n is an alphabetic character:

As usual, we write C for the random variable and c for an actual value. Technically speaking, C is a function that returns an infinite sequence, and we ought to define it as a lazy list rather than writing out a non-terminating while loop.

```
alphabet = ['a', 'b', ...]   # all possible characters incl. punctuation
next_char_prob = {('a','a'): [0,0,.1,...], ('a','b'): [.5,0,...]}
c = ['o', 'n']  # arbitrary starting string of length 2

while True:
    p = next_char_prob[(text[-2],text[-1])]  # the last two elements
    c.append(random.choice(alphabet, weights=p))
```

In this code, next_char_prob is a dictionary where each value p=next_char_prob[...] is a vector of probabilities, and where p[i] is the probability that the next character is alphabet[i].

We can measure next_char_prob for a piece of literature by looking at all trigrams i.e. sequences of three characters. Markov tabulated m-grams for several works by famous Russian authors, and suggested that the next_char_prob table might be used to identify an author.

Here is some Shakespeare generated in this method. The source is all of Shakespear's plays, with stage directions omitted, and converted to lowercase.

> *once. sen thery lost like kin ancry on; at froan, is ther page: good haves have emst upp'd ne kining, whows th lostruck-ace. 'llycur wer; hat behit mord. misbur greake, weave o'er, thousing i se to; ang shal spird*

Here is some text generated with 5-grams rather than trigrams.

> *once is pleasurely. though the the with them with comes in hand. good. give and she story tongue. what it light, would in him much, behold of busin! how of ever to yearling with then, for he more riots annot know well.*

Definition. A *Markov chain* is a sequence $(X_0, X_1, X_2, \dots)$ where each X_n is a discrete random variable and

$$\mathbb{P}(X_{n+1} = x_{n+1} \mid X_0 = x_0, X_1 = x_1, \dots, X_n = x_n) = \mathbb{P}(X_{n+1} = x_{n+1} \mid X_n = x_n) \quad \text{for all } x_0, \dots, x_{n+1}. \tag{15}$$

(To be precise, the equation must hold for all $x_0, \ldots, x_{n+1}$ such that $\mathbb{P}(X_0 = x_0, \ldots, X_n = x_n) > 0$, since otherwise the conditional probability isn't defined.) If equation (15) holds and furthermore the probability does not depend on n, i.e. if there is some matrix P such that

$$\mathbb{P}(X_{n+1} = y \mid X_n = x) = P_{xy}$$

then the process is called a *time-homogeneous Markov chain* with *transition matrix* P.

In IA *Foundations of Computer Science* you learnt about finite automata. What is the relationship to Markov chains?

- Finite automata and Markov chains both have a set of possible states, and a lookup table / transition relation that describes progression from one state to the next.
- Finite automata are for describing algorithms that accept input, so the lookup table specifies 'what happens next, based on the current state and the given input symbol?' Markov chains are for describing systems that evolve by themselves, without input.
- Non-deterministic finite automata allow there to be several transitions out of a state, but they do not specify the probability of each transition, since they are intended to model 'what are all the things that might happen?' Markov chains do specify the transition probabilities, since they are intended to model 'what are the things that typically happen?'
- Markov chains are allowed to have an infinite state space, e.g. the space of all integers. (They can even be defined with uncountable state spaces in which case X_n is a continuous random variable; the definition needs to be modified to refer to transition density functions rather than transition probabilities.)

The word *chain* means that the sequence $(X_n)_{n \geq 0}$ is indexed by an integer n. There are related definitions for continuous-time processes, and these will be used in Part II *Computer Systems Modelling*, but we will not study them further in this course.

In the Shakespeare example, the next character was chosen based on the previous *two* characters, which at first glance looks like it doesn't satisfy equation (15). The trick is to define X appropriately: in this case we should define $X_n = (C_n, C_{n+1})$. Then, the text generation rule can be rewritten as

```
x = [('o', 'n')]  # arbitrary value for x[0]
c = x[0]

while True:
    lastx = x[-1]
    nextchar = random.choice(alphabet, next_char_prob[lastx])
    nextx = (lastx[-1], nextchar)
    x.append(nextx)
    c.append(nextchar)
```

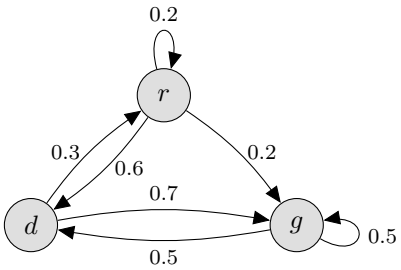
This way of writing the code makes it clear that X is a time-homogeneous Markov chain. The actual text C is a byproduct of X.

CALCULATIONS BASED ON MEMORYLESSNESS

Equation (15) says that if we know the present state X_n, then the past gives us no extra information about what will happen next. This is known as *memorylessness*, or as the *Markov property*. Most calculations with Markov chains revolve around conditioning on a previous step and then applying memorylessness. Look back at Section 1.2 to see how we we used memorylessness in our calculations for the Bitcoin example. Here are some more examples.

Example 4.2 (Multistep transition probabilities). The winter weather in Cambridge varies from grey (g) to drizzle (d) to rain (r). Suppose that the weather changes from day to day according to the Markov chain drawn below. If it is overcast today, what's the chance that

it will be overcast three days from today?



$$P = \begin{array}{c} \\ r \\ d \\ g \end{array} \begin{array}{c} \begin{array}{ccc} r & d & g \end{array} \\ \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.3 & 0 & 0.7 \\ 0 & 0.5 & 0.5 \end{pmatrix} \end{array}$$

The state transition diagram can also be written as a matrix P of transition probabilities. When you write out a Markov transition diagram or matrix, double-check that every row sums to 1, i.e. that all the total probability of all edges out of a node is equal to 1.

The question is asking us to calculate

$$\mathbb{P}(X_3 = g \mid X_0 = g).$$

The most useful strategy for questions like this is: think of the causal diagram that underlies the model, and use the law of total probability to add in variables for all the missing nodes. In the case of Markov chains, the fundamental causal process is 'choose the next state based on the current state', which we could draw as

$$X_0 \to X_1 \to X_2 \to X_3 \to \dots$$

so let's add in variables for the missing variables X_1 and X_2:

$$\mathbb{P}(X_3 = g \mid X_0 = g) = \sum_{x_1,\, x_2} \mathbb{P}(X_3 = g, X_2 = x_2, X_1 = x_1 \mid X_0 = g).$$

There are two quite separate diagrams involved here: the state space diagram which shows transition probabilities between states; and the causal diagram which shows which random variables depend on which other random variables.

The next most useful strategy is: think of the causal diagram that underlies the model, and rewrite conditional probabilities so that they are of the form 'probability of a variable for one node in the causal diagram, given its preceding nodes'. So, let's try to rewrite the probability we're after in terms of $\mathbb{P}(X_3 = g \mid X_2 = x_2)$ etc.

This approach, of drawing a causal diagram and rewriting conditional probabilities so they respect the edges, is the essence of Bayes' rule and indeed of all of Bayesian modelling.

$$\begin{aligned}
&\mathbb{P}(X_3 = g, X_2 = x_2, X_1 = x_1 \mid X_0 = g) \\
&\quad = \frac{\mathbb{P}(X_3 = g, X_2 = x_2, X_1 = x_1, X_0 = g)}{\mathbb{P}(X_0 = g)} \text{ from defn. of conditional probability} \\
&\quad = \frac{\mathbb{P}(X_3 = g \mid X_2 = x_2, X_1 = x_1, X_0 = g)\, \mathbb{P}(X_2 = x_2, X_1 = x_1, X_0 = g)}{\mathbb{P}(X_0 = g)} \\
&\quad = \frac{\mathbb{P}(X_3 = g \mid X_2 = x_2)\, \mathbb{P}(X_2 = x_2, X_1 = x_1, X_0 = g)}{\mathbb{P}(X_0 = g)} \text{ by the Markov property} \\
&\quad = \frac{\mathbb{P}(X_2 = x_2, X_1 = x_1, X_0 = g)}{\mathbb{P}(X_0 = g)} P_{x_2 g} \text{ since the transition matrix is } P \\
&\quad \vdots \\
&\quad = P_{g\,x_1} P_{x_1\,x_2} P_{x_2\,g}.
\end{aligned}$$

We have thus derived an explicit formula for the probability we want,

$$\mathbb{P}(X_3 = g \mid X_0 = g) = \sum_{x_1, x_2} P_{g x_1} P_{x_1 x_2} P_{x_2 g}$$

which may be written in matrix form as $[P^3]_{gg}$. To compute it in Python,

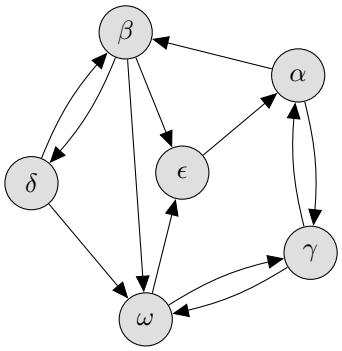
```
P = np.array([[0.2, 0.6, 0.2], [0.3, 0, 0.7], [0, 0.5, 0.5]])
assert all(P.sum(axis=1) == 1)      # check row sums are all equal to 1
(P @ P @ P)[2,2]                    # compute P^3 then pick out element at [2,2]
np.linalg.matrix_power(P, 3)[2,2]   # another way to compute P^3
```

Exercise 4.3 (Extended Markov property). Let X be a Markov chain. The Markov property, equation (15) says that if we know the present state X_n, then the past $(X_0, \dots, X_{n-1})$ gives

us no extra information about the next step X_{n+1}. Prove that the same holds true further into the future, i.e. for any $(x_0, \dots, x_{n+m})$,

$$\begin{aligned}\mathbb{P}(X_{n+m} = x_{n+m}, \dots, X_{n+1} = x_{n+1} \mid X_n = x_n, \dots, X_0 = x_0) \\ = \mathbb{P}(X_{n+m} = x_{n+m}, \dots, X_{n+1} = x_{n+1} \mid X_n = x_n).\end{aligned}$$

Example 4.4 (Hitting probabilities). A web surfer starts at page α, and from each page picks an outgoing link at random from that page. What is the chance they hit ω before returning to α?



Let X_n be the page that the web surfer is on after n clicks, $X_0 = \alpha$, and write X for the entire process $X = (X_n)_{n \geq 0}$. We want to calculate

$$\mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_0 = \alpha\right).$$

This is open-ended—X could first hit those two destinations at any $n \geq 1$—so there's no clean way for us to condition on the entire path, as we did in Example 4.2. Instead, let's condition just on X_1:

$$\begin{aligned}&\mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_0 = \alpha\right) \\ &\quad = \sum_{x_1} \mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_1 = x_1,\, X_0 = \alpha\right) \mathbb{P}\big(X_1 = x_1 \mid X_0 = \alpha\big). \qquad (16)\end{aligned}$$

Here we have used a conditional-probability version of the law of total probability. The regular version is

$$\mathbb{P}(A) = \mathbb{P}(A \mid C)\,\mathbb{P}(C) + \mathbb{P}(A \mid \bar{C})\,\mathbb{P}(\bar{C})$$

and the conditional-probability version can be proved in exactly the same way:

$$\begin{aligned}\mathbb{P}(A \mid B) &= \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} = \frac{\mathbb{P}(A \cap B \cap C)}{\mathbb{P}(B)} + \frac{\mathbb{P}(A \cap B \cap \bar{C})}{\mathbb{P}(B)} \\ &= \mathbb{P}(A \mid B \cap C)\frac{\mathbb{P}(B \cap C)}{\mathbb{P}(B)} + \mathbb{P}(A \mid B \cap \bar{C})\frac{\mathbb{P}(B \cap \bar{C})}{\mathbb{P}(B)} \\ &= \mathbb{P}(A \mid C \cap B)\,\mathbb{P}(C \mid B) + \mathbb{P}(A \mid \bar{C} \cap B)\,\mathbb{P}(\bar{C} \mid B).\end{aligned}$$

In equation (16), the first term involves conditioning on both X_0 and X_1, and the extended Markov property (Example 4.3) says that conditional on X_1 the future is independent of X_0. Thus

$$\mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_0 = \alpha\right) = \sum_{x_1} P_{\alpha x_1}\, \mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_1 = x_1\right).$$

The final trick is to 'reset the clock'. Let's define

$$\pi_x = \mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 0 \\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_0 = x\right).$$

It doesn't make any difference whether we start measuring time from $n = 0$ or from $n = 1$, since the process follows the same dynamics regardless, thus

$$\mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha \end{array} \;\middle|\; X_0 = \alpha\right) = \sum_x P_{\alpha x}\pi_x \tag{17}$$

and furthermore (by repeating the entire conditioning argument we have just been through)

$$\pi_x = \sum_y P_{xy}\pi_y \tag{18}$$

and clearly $\pi_\alpha = 0$ and $\pi_\omega = 1$. Rewriting equations (17) and (18) as matrix equations, π is a 6-dimensional vector with elements in $[0, 1]$ and

$$\mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha \end{array} \;\middle|\; X_0 = \alpha\right) = \big[P\pi\big]_\alpha, \quad \pi = P\pi.$$

In Python,

```
# States are in the order [α,β,γ,δ,ϵ,ω]
# Set up an adjacency matrix for the graph,
# then scale it so rows sum to 1 (and assert they do)
P = np.array([[0,1,1,0,0,0], [0,0,0,1,1,0], [1,0,0,0,0,1],
              [0,1,0,0,0,1], [1,0,0,0,0,0], [0,0,1,0,1,0]])
P = P / P.sum(axis=1)[:, np.newaxis]
assert all(P.sum(axis=1) == 1)
# We want to solve P.π=π i.e. (P-I).π=0, and also π[0]=0 and π[5]=1
# Bundle all the equations together in a matrix, and solve with np.linalg.lstsq
A = np.concatenate((P - np.eye(6), [[1,0,0,0,0,0], [0,0,0,0,0,1]]))
π = np.linalg.lstsq(A, [0,0,0,0,0,0, 0,1])[0]
# Return the hitting probability we wanted to calculate
(P @ pi)[0]
```

MEMORY LENGTH

The rule we used to generate pseudo-Shakespeare, "pick the next charactacter based on the preceding m", produces better-looking results for larger m—but the larger m is, the more storage space we need for the lookup table, and the fewer $(m + 1)$-grams we have with which to estimate frequencies. If m gets even larger, the algorithm can't do much more than regurgitate the input text on which it was trained.

Neural networks can be used to get around these limitations: they can learn how much information from preceding elements in the sequence should be incorporated into the state of the Markov chain, and they're not limited to fixed-m state descriptor. Here is an example of Shakespeare generated using a neural network rather than trigram frequencies[31].

> *PANDARUS:*
> *Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.*
>
> *Second Senator:*
> *They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.*
>
> *DUKE VINCENTIO:*
> *Well, your wit is in the care of side and that.*

[31] Andrej Karpathy, *The unreasonable effectiveness of recurrent neural networks*, May 2015, `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. He writes "There's something magical about Recurrent Neural Networks (RNNs) ... We'll train RNNs to generate text character by character and ponder the question 'how is that even possible?' "

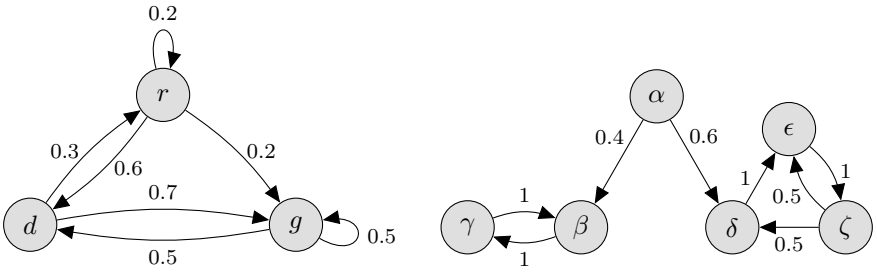
4.2. Estimation in a hidden Markov model

Example. A person is moving about. At each timestep, we receive a noisy GPS reading of their current location. How can we estimate their current location?

This is the topic of Example Sheet 3.

4.3. Limit theorems

When analysing Markov chains, it's often useful to be able to ask about their long-run average behaviour. We asked the same question in Section 2 about sums of independent random variables. Markov chains however have a richer range of possible behaviours, and it turns out there are three separate ways to ask 'what is average behaviour?' To reduce the mathematical overhead we will restrict attention in this section to time-homogeneous Markov chains with a finite state space (though most of the results also hold for infinite state spaces). We will illustrate with two examples, the Markov chain for Cambridge weather from Section 4.1, and a pathological case.



4.3.1. STATIONARY BEHAVIOUR

Definition. A Markov chain is said to be *stationary* if its distribution does not change over time, i.e. if there is a vector π such that $\mathbb{P}(X_n = x) = \pi_x$ for all n. Then π is called the *stationary distribution*, and also the *equilibrium distribution*.

The word 'stationary' does not mean that the Markov chain has somehow stopped—a Markov chain is defined to go on forever, always stepping randomly from state to state. It is the *distribution* that is stationary i.e. unchanging.

We can compute a stationary distribution using the same sort of calculations based on memorylessness that we used in Section 4.1. Take the Cambridge weather Markov chain, for example. If π is a stationary distribution then, for any n,

$$\pi_x = \mathbb{P}(X_n = x) = \sum_y \mathbb{P}(X_n = x \mid X_{n-1} = y)\,\mathbb{P}(X_{n-1} = y) = \sum_y \pi_y P_{yx} \tag{19}$$

where P is the transition matrix. In matrix notation, $\pi = \pi P$ or equivalently $(P - I)^\top \pi = 0$. To pin down π completely we need an extra equation, $\sum_x \pi_x = 1$; this equation must hold because π is a distribution, and it's necessary because otherwise we could multiply π by a constant and still have a solution to $(P - I)^\top \pi = 0$. In Python,

```
P = np.array([[0.2, 0.6, 0.2], [0.3, 0, 0.7], [0, 0.5, 0.5]])
A = np.concatenate(((P - np.eye(3)).transpose(), [[1,1,1]]))
π = np.linalg.lstsq(A, [0,0,0, 1])[0]
```

We can compute a stationary distribution for the pathological Markov chain using exactly the same method, but there is a problem: equation (19) has multiple solutions, even after imposing the extra equation $\sum_x \pi_x = 1$. If we just write out all the equations longhand,

$$\begin{aligned}
\pi_\alpha &= 0 \\
\pi_\beta &= 0.4\pi_\alpha + \pi_\gamma \\
\pi_\gamma &= \pi_\beta \\
\pi_\delta &= 0.6\pi_\alpha + 0.5\pi_\zeta \\
\pi_\epsilon &= \pi_\delta + 0.5\pi_\zeta \\
\pi_\zeta &= \pi_\epsilon \\
\pi_\alpha + \pi_\beta + \pi_\gamma + \pi_\delta + \pi_\epsilon + \pi_\zeta &= 1
\end{aligned}$$

and solve these equations simultaneously, we discover that the general solution for $\pi = [\pi_\alpha, \pi_\beta, \pi_\gamma, \pi_\delta, \pi_\epsilon, \pi_\zeta]$ is

$$\pi = a\left[0, 1/2, 1/2, 0, 0, 0\right] + (1-a)\left[0, 0, 0, 1/5, 2/5, 2/5\right] \tag{20}$$

for any real value a (though only $a \in [0, 1]$ will yield a legitimate probability distribution). In Python, if we look carefully at the output of np.linalg.lstsq() and read the documentation, we see it telling us that the linear equation does not have a unique solution; there are further np.linalg tools that can extract the general form of the solution.

Equation (20) actually has a nice intuitive explanation. The Markov chain could be spending all its time in states $\{\beta, \gamma\}$ with stationary distribution $[1/2, 1/2]$, or it could be spending all its time in states $\{\delta, \epsilon, \zeta\}$ with stationary distribution $[1/5, 2/5, 2/5]$.

Theorem (uniqueness of stationary distribution). Consider a Markov chain with transition matrix P and a finite state space. The Markov chain is called *irreducible* if it is possible to get from any state to any other. If the Markov chain is irreducible, then there is a unique stationary distribution, and it is the unique solution π to

$$\pi = \pi P, \quad \pi^\top 1 = 1. \tag{21}$$

This theorem does not say that the chain itself is stationary, only that it has a stationary distribution. In other words, if we pick the initial state X_0 randomly according to π, then X_1 will have distribution π and so will X_2 and so on. But if we pick the initial state in some other way, e.g. we start at $X_0 = g$ in the Cambridge weather example, this theorem doesn't tell us what will happen.

Example. The Cambridge weather Markov chain can get from any state to any other state; to get from g to r takes two steps, and all the others can be achieved in one step. Therefore it is irreducible, therefore it has a unique stationary distribution.

The pathological Markov chain is not irreducible, because it is impossible to get from β to α.

4.3.2. DETAILED BALANCE

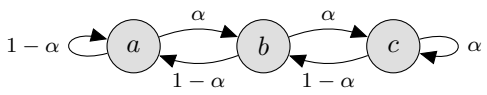
Often, when we want to find the stationary distribution, there's nothing for it but to use np.linalg and solve a matrix equation. In some special cases the Markov chain has a form that lets us find the stationary distribution with very little algebra. This seems like a curiosity, not worth mentioning in a data science course—except that there is a clever trick for generating random variables from general Bayesian posterior distributions that relies on exactly this special case. The clever trick is called Gibbs sampling, and it is taught in Part II *Machine Learning and Bayesian Inference*.

Theorem (detailed balance). Let X be an irreducible Markov chain with transition matrix P. If there exists a vector π such that $0 \le \pi \le 1$ and $\sum_x \pi_x = 1$ and

$$\pi_x P_{x\,y} = \pi_y P_{y\,x} \quad \text{for all states } x \text{ and } y \tag{22}$$

then π is the stationary distribution (which we know is unique, since the chain is irreducible). Equation (22) is called the *detailed balance* condition.

Exercise 4.5. Calculate the stationary distribution of the Markov chain



Is it irreducible? Yes, it's easy to see that there's a path from any state to any other. Therefore there is a unique stationary distribution. It never hurts to try to solve the detailed balance equations; either we find the stationary distribution without much work, or we quickly discover that they can't be solved and we have to solve the full equations (21). In this case, the detailed balance equations are

$$\begin{aligned}
&\text{for } (a,b) \text{ and } (b,a): && \pi_a \alpha = \pi_b (1-\alpha) \\
&\text{for } (a,c) \text{ and } (c,a): && \pi_a 0 = \pi_c 0 \\
&\text{for } (b,c) \text{ and } (c,b): && \pi_b \alpha = \pi_c (1-\alpha) \\
&\text{for } (a,a) \text{ etc.:} && \pi_a (1-\alpha) = \pi_a (1-\alpha) \text{ etc.}
\end{aligned}$$

and they have the solution

$$\pi_b = \pi_a \frac{\alpha}{1-\alpha}, \quad \pi_c = \pi_a \Big(\frac{\alpha}{1-\alpha}\Big)^2.$$

Putting in the constraint $\pi_a + \pi_b + \pi_c = 1$, we get

$$\big[\pi_a, \pi_b, \pi_c\big] = \frac{1}{1 + \alpha/(1-\alpha) + \alpha^2/(1-\alpha)^2} \Big[1, \frac{\alpha}{1-\alpha}, \Big(\frac{\alpha}{1-\alpha}\Big)^2\Big].$$

Exercise 4.6 (Random walk on an undirected graph). A knight moves on an otherwise empty chessboard, each timestep picking one of its legal moves at random (out of 8 legal moves if it is in the center of the board, and 2 legal moves if it is in a corner). Show that the stationary probability of being in position x is $m_x/336$, where m_x is the number of legal moves out of position x.

We should first check whether the Markov chain described in the question is irreducible, since otherwise there isn't even a unique stationary distribution. This is just a matter of sketching a chessboard and persuading ourselves that a knight can indeed get from any position to any other position, given enough moves.

The question tells us the stationary distribution and asks us to verify it. We could plug it into the full equations (21), but if it happens to solve the detailed balance equations then that is sufficient and our work will be simpler. The detailed balance equations are

$$\begin{aligned} \frac{m_x}{336} \times \frac{1}{m_x} &= \frac{m_y}{336} \times \frac{1}{m_y} \quad \text{if } x \leftrightarrow y \text{ is legal,} \\ \frac{m_x}{336} \times 0 &= \frac{m_y}{336} \times 0 \quad \text{if } x \leftrightarrow y \text{ is illegal.} \end{aligned}$$

These equations are certainly true, and they are the only equations that need to be satisfied, since $x \to y$ is legal if and only if $y \to x$ is legal. Therefore the suggested distribution solves detailed balance.

Finally, we need to verify that the suggested distribution is indeed a distribution, i.e. that it sums to 1. Counting the number of possible moves from every position on the chessboard gives a total of 336, thus $\sum_x m_x/336 = 1$.

It's easy to check that the result described here can be generalised to a random walk on any undirected graph.

4.3.3. ERGODIC THEOREM

Theorem (ergodicity). Let X be an irreducible Markov chain with stationary distribution π. Then the long-run average of time spent in each state converges to π. Mathematically,

$$\mathbb{E}\Big(\frac{1}{n}\sum_{i=1}^{n} 1_{X_i = x}\Big) \to \pi_x \quad \text{as } n \to \infty, \text{ for all states } x. \tag{23}$$

If the Markov chain's initial state X_0 were chosen from distribution π, then we know from Section 4.3.1 that X_n would have distribution π for every n, thus $\mathbb{E}1_{X_i=x} = \mathbb{P}(X_i = x) = \pi_x$ for all i, and so (23) would be true exactly, no need for a limit. What's remarkable is that the theorem holds regardless of how the initial state is chosen.

Example. Consider the pathological Markov chain, starting at $X_0 = \alpha$. This chain is not irreducible, so the ergodic theorem doesn't apply directly. But we can still say how the chain behaves: with probability 0.4 it jumps to $X_1 = \beta$, and thereafter it behaves just like an irreducible chain on $\{\beta, \gamma\}$ and spends half its time in each of those two states; or with probability 0.6 it jumps to $X_1 = \delta$, and thereafter it behaves just like an irreducible chain on $\{\delta, \epsilon, \zeta\}$ and spends roughly 20% of its time in δ, 40% in ε, and 40% in ζ.

4.3.4. LIMITING BEHAVIOUR

In the Cambridge weather Markov chain, the ergodic theorem tells us that the long-run fraction of rainy days is equal to π_r, where π is the stationary distribution. So we'd expect

that, if we pick a day arbitrarily, the probability of rain is π_r. This is works for the Cambridge weather example, but it doesn't always work... the caveat is illustrated by states β and γ in the pathological Markov chain: if the chain starts in $X_0 = \beta$ then $X_n = \beta$ for even n and $X_n = \gamma$ for odd n, and so we can't make a blanket claim about 'typical X_n'. The following theorem gives a general condition under which time-averages correspond to typical values.

Theorem. Let X be a Markov chain. A state x is said to be *periodic* if there exists an n_0 such that $\mathbb{P}(X_n = x \mid X_0 = x) > 0$ for all $n \geq n_0$. If the chain is irreducible and has an aperiodic state, then all its states are aperiodic, and furthermore

$$\mathbb{P}(X_n = x \mid X_0 = y) \to \pi_x \quad \text{as } n \to \infty, \text{ for all states } x.$$

Note that $\mathbb{P}(X_n = x \mid X_0 = y) = [P^n]_{x\,y}$ where P is the transition matrix, according to our calculations in Example 4.2.