Part IV
Advanced probability modelling

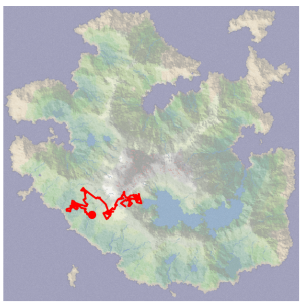
9. Probability models for systems

There are many datasets for which, if we are to describe them usefully, we need a *system* of random variables—a collection of random variables that are related to each other. For example,

- Suppose we pick a random 6-word sentence from Shakespeare, "Confusion now hath made his masterpiece!" This is rather more poetic than the sentence we get from picking 6 words at random, "fortunes blood now between make the". If we let the words be $(X_1, \ldots, X_6)$, then clearly we should not treat them as independent.
- Science is often concerned with the laws that describe how a system changes over time, such as Newton's laws of motion. We might use probabilistic laws to describe how a system changes; almost every interesting simulation works like this. If X_n is the state of the system at timestep n, then the sequence $(X_0, X_1, \ldots)$ is a random system, in which each state depends on the preceding state(s).
- Even the simple Gaussian mixture model from section 1.5 is a random system. We considered a system made of two random variables, K and X: first pick a random cluster K, then generate a Normal random variable X whose mean and variance are chosen based on K.

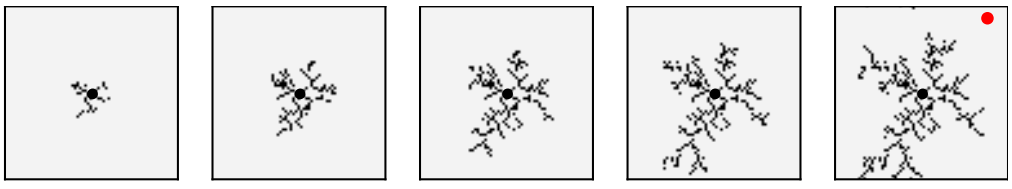
Here are some more illustrations.

Example 9.1 (Random walk; location tracking).
Consider an object moving around in space, for example a stoat in the wild. Let X_n be its location at timestep n. We might model this with a simulator that generates a new random location X_{n+1} as a function of its current location X_n. Or we might use a more elaborate model which includes velocity, or mode (slinking, gambolling, chasing, ...). The sequence $(X_0, X_1, \ldots)$ is called a *random walk*.



Now, suppose we release the animal with a GPS tracker and a low-resolution camera—and then the GPS tracker breaks, but we still have a feed from the camera, and we know roughly what terrain it was in each day (the bottom panel in the plot above). If we know the map that it's moving on, and we know the terrain it sees each day, how can we estimate its current location?

Example 9.2 (Probabilistic automata).
Every computer scientist knows about Conway's Game of Life. That game has deterministic rules. What about automata with probabilistic rules? Here's an example, a Snowflake Simulator.



We start with a seed at the center of the image, and a particle placed randomly on the boundary. The particle moves randomly until it touches the seed, at which point it attaches. Start a new particle on the boundary, and repeat.

There are two levels of random system here. At the lower level, we can take the snowflake that's accumulated so far as fixed, and model the particle's movement as a random walk. A typical question here: rather than simulating the particle step-by-step, couldn't we just run a single computation to calculate the chance of every attachment point? This would significantly speed up the simulation.

At the higher level, we might be interested in the growth of the snowflake, i.e. in the sequence $(X_0, X_1, X_2, \dots)$ where X_n is the state after n particles have attached themselves. Or we might be interested in the shape of a snowflake, i.e. in the matrix $[X_n^{ij}]_{1 \le i \le w, 1 \le j \le h}$ where X_n^{ij} is the pixel value at coordinate (i, j) in the image of the snowflake after n particles have attached themselves. What's the fractal dimension of this image? What's its diameter? These are hugely challenging questions to answer.

Example 9.3.

The Russian mathematician Andrei Markov (1856–1922) invented a new type of random process, 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:

```
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[(c[-2],c[-1])]  # lookup based on the last two elements
    nextchar = random.choice(alphabet, weights=p)
    c.append(nextchar)
```

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.*

And here is an example of Shakespeare generated using a recurrent neural network, which is a souped-up version of m-gram frequencies[35]

> *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.*

.

* ✻ *

In Part II, you will come across random systems 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 numerically computing posterior distributions
- in *Data Science Principles and Practice* they are used in recurrent neural networks, useful for language modelling
- in *Information Theory* they are used to describe noisy communications channels, and also the data streams sent over such channels

[35] 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?' "

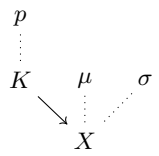
9.1. Causal diagrams

*** TODO: Latent generative models

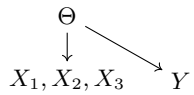
When faced with a bewildering wall of code or text describing a random system, it may be useful to draw out the *causal diagram*[36]. This is a directed acyclic graph with a node for every random variable, and arrows indicating which random variables are generated from which other variables. It's also a good idea to draw on any other unknown parameters, and to label them differently[37] to the random variables. Drawing causal diagrams is good mental hygiene, an easy way to see if we've understood the problem description. Here are some examples.

```
p = [0.085, 0.278, 0.637]
μ = [9709, 19822, 22756]
σ = [422, 559, 3382]
k = np.random.choice(range(3), p=p)
x = np.random.normal(loc=μ[k], scale=σ[k])
```


```
def rgmm(p, μ, σ):
    k = np.random.choice(range(len(p)), p=p)
    x = np.random.normal(loc=μ[k], scale=σ[k])
    return x
```




```
θ = np.random.beta(2,3)
x1,x2,x3 = np.random.binomial(n=10, p=θ, size=3)
y = np.random.binomial(n=1, p=θ)
```

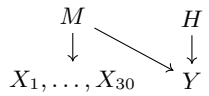



```
x = 0
while True:
    yield x
    x = x + np.random.choice([-1,1])
```

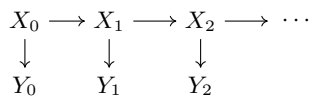
$X_0 \to X_1 \to X_2 \to \cdots$

I am prototyping a diagnostic test for a disease. In healthy patients, the test result is Normal$(0, 2.1^2)$. In sick patients it is Normal$(\mu, 3.2^2)$, but I have not yet established a firm value for μ. My prior for μ is $M \sim$ Normal$(5, 3^2)$.

I trialled the test on 30 patients whom I know to be sick, and the mean test result was 10.3. I subsequently apply the test to a new patient, and get the answer 8.8. I wish to know whether this new patient is healthy or sick. Let $h \in \{\text{healthy}, \text{sick}\}$ be the status of the new patient, and use the prior distribution $\Pr_H(\text{healthy}) = 0.99, \Pr_H(\text{sick}) = 0.01$.



Consider a moving object, from which I get noisy GPS readings. Let X_n be the location at timestep n, and let Y_n be the reading.




```
x = 0
while True:
    y = x + np.random.normal(0,1)
    yield y
    x = x + np.random.choice([-1,1])
```

[36] Various fields of study have their own conventions about diagrams of the general sort described here. In computer science there are *data-flow diagrams* to describe a multi-step process for manipulating data. Bayesian modelling uses *graphical models*.

The term we are using here, *causal diagram*, is used by Judea Pearl, and it has two extra connotations. First, that the graph is a directed acyclic graph, i.e. it has no loops; this implies that the graph can be serialized as lines of code to be run one after another. Second, that the arrows are causal links: if we intervene to change one of the variables, for example by setting it manually inside an interactive debugger, then the rest of the diagram is not affected.

[37] Bayesianists claim there is no such thing: they say that all unknown parameters should be treated as random variables.

CALCULATIONS WITH CAUSAL DIAGRAMS

When we're trying to calculate a probability, the causal diagram can steer us in the right direction. There are two broad strategies, (1) rearrange any probabilities to be in the form 'probability of a child node, conditional on its parents', (2) if the parents aren't already in the expression, find a way to incorporate them.

The algebraic tactics we use to achieve these strategic goals are the basic rules of probability. The table below shows, on the left, the basic rules of elementary probability. On the right it shows the *conditional form*—all the rules still work if we just stick 'conditional on C' onto every term, for any event C with $\mathbb{P}(C) > 0$. Don't think of these as separate rules, just think of them as "the basic rules of probability still work, even if there's baggage being carried around".

Conditional probability:

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A,B)}{\mathbb{P}(B)} \text{ if } \mathbb{P}(B) > 0 \qquad \mathbb{P}(A \mid B, C) = \frac{\mathbb{P}(A,B \mid C)}{\mathbb{P}(B \mid C)} \text{ if } \mathbb{P}(B \mid C) > 0$$

Independence of A and B; and conditional independence of A and B given C:

$$\mathbb{P}(A,B) = \mathbb{P}(A)\,\mathbb{P}(B) \qquad \mathbb{P}(A,B \mid C) = \mathbb{P}(A \mid C)\,\mathbb{P}(B \mid C)$$
$$\mathbb{P}(A \mid B) = \mathbb{P}(A) \qquad \mathbb{P}(A \mid B, C) = \mathbb{P}(A \mid C)$$

Sum rule, and the law of total probability, for events $\{B_1, B_2, \dots\}$ that partition the sample space:

$$\mathbb{P}(A) = \sum_i \mathbb{P}(A, B_i) \qquad \mathbb{P}(A \mid C) = \sum_i \mathbb{P}(A, B_i \mid C)$$
$$= \sum_i \mathbb{P}(A \mid B_i)\,\mathbb{P}(B_i) \qquad = \sum_i \mathbb{P}(A \mid B_i, C)\,\mathbb{P}(B_i \mid C)$$

Bayes's rule:

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(B \mid A)\,\mathbb{P}(A)}{\mathbb{P}(B)} \qquad \mathbb{P}(A \mid B, C) = \frac{\mathbb{P}(B \mid A, C)\,\mathbb{P}(A \mid C)}{\mathbb{P}(B \mid C)}$$

Each of these rules has a corresponding version for random variables. For example, the law of total probability turns into

$$\Pr_X(x) = \int_y \Pr_X(x \mid Y = y)\,\Pr_Y(y)\,dy$$
$$\Pr_X(x \mid C) = \int_y \Pr_X(x \mid Y = y, C)\,\Pr_Y(y \mid C)\,dy$$
$$\Pr_X(x \mid Z = z) = \int_y \Pr_X(x \mid Y = y, Z = z)\,\Pr_Y(y \mid Z = z)\,dy$$

Again, don't think of these as separate rules, just think of them as "the way to write the rules of basic probability, so that they work with continuous random variables". The first version, for $\Pr_X(x)$, tells us something useful even when $\mathbb{P}(X = x) = 0$ or $\mathbb{P}(Y = y) = 0$. The second version, for $\Pr_X(x \mid C)$, is another instance of "all the rules still work if we stick 'conditional on C' on, as long as $\mathbb{P}(C) > 0$". The third, for $\Pr_X(x \mid Z = z)$, is what we need if we want to condition on $C = \{Z = z\}$, an event that might have probability zero.

Conditional independence and causal diagrams. The most important of all of these rules is conditional independence. When we see code, or a causal diagram, like

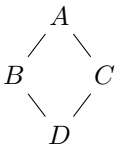
```
x=np.random.uniform(); y=f(x); z=g(y)
```
or $X \to Y \to Z$

we mean "if you want to know about Z, and you know Y, then knowing X doesn't give you any further information". In other words, Z and X are conditionally independent given Y,

$$\Pr_Z(z \mid Y = y, X = x) = \Pr_Z(z \mid Y = y).$$

More generally, at any node Z in a causal diagram, if we condition on all of its immediate parents Y, that node is conditionally independent from everything else in the diagram, apart from its descendants.

ILLUSTRATIONS



What is $\Pr(a, b, c, d)$? The general strategy is "rearrange any probabilities to be in the form: probability of a child node, conditional on its parents". Applying this first to d,

$$\begin{aligned}\Pr(a, b, c, d) &= \Pr(d \mid a, b, c) \Pr(a, b, c) \quad \text{defn. of cond. prob} \\ &= \Pr(d \mid b, c) \Pr(a, b, c) \quad \text{cond. indep. of } d \text{ and ancestors.}\end{aligned}$$

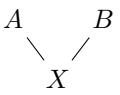
Continuing in this way,

$$\Pr(a, b, c, d) = \Pr(d \mid b, c)\ \Pr(b \mid a)\ \Pr(c \mid a)\ \Pr(a).$$

What is $\Pr_X(x)$? The general strategy is "if the parents aren't already in the expression, find a way to incorporate them." We can achieve this using the law of total probability:

$$\Pr_X(x) = \sum_k \Pr_X(x \mid K = k) \Pr_K(k).$$

This is how we derived the density function for the Gaussian mixture model, introduced in section 1.5 and mock exam question 1.



What is $\Pr_X(x \mid A = a)$? The general strategy is "if the parents aren't already in the expression, find a way to incorporate them." We already have one parent, A, and we want the other. We can achieve this using the law of total probability, in its conditional form, carrying along the baggage $\{A = a\}$.

$$\begin{aligned}\Pr_X(x \mid A = a) &= \int_b \Pr_X(x \mid B = b, A = a)\ \Pr_B(b \mid A = a)\, db \quad \text{law tot. prob.} \\ &= \int_b \Pr_X(x \mid B = b, A = a)\ \Pr_B(b)\, db \quad \text{ind. of } B \text{ and } A\end{aligned}$$

The causal diagram has no paths from A to B or vice versa, so A and B are independent.

What is $\Pr_K(k \mid X = x)$? The general strategy is "rearrange any probabilities to be in the form: probability of a child node, conditional on its parents". We're asked for $\Pr_K(k \mid X = x)$, but we want to flip it around, and Bayes's rule is the tool.

$$\Pr_K(k \mid X = x) = \kappa \Pr_X(x \mid K = k) \Pr_K(k)$$

for a suitable constant κ, chosen to make the expression sum to one. See exercise sheet 2 question 2.

9.2. Markov chains and memorylessness

tl;dr.A *Markov chain* is a sequence $(X_0, X_1, X_2, \dots)$ where each X_{n+1} is a discrete random variable, generated randomly according to the causal diagram

$$X_0 \to X_1 \to X_2 \to \cdots$$

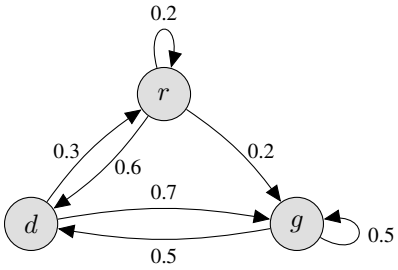
and where X_{n+1} is generated from X_n according to

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

for some matrix P, called the *transition matrix*. The *state space* of the Markov chain is the set of values from which the X_n are drawn, and it is assumed to be countable.

The causal diagram tells us that, if we know the current state X_n, then the past $(X_m)_{m<n}$ is irrelevant for working out the probabilities of future events, which are based on $(X_m)_{m>n}$. This is called *memorylessness*.

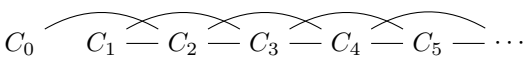
Example 9.4 (Markov model for Cambridge weather).
The winter weather in Cambridge varies from grey (g) to drizzle (d) to rain (r). Suppose that the weather changes from day to day according to a time-homogenous Markov chain. We can show it either with the transition matrix, or with the corresponding *state space diagram*.



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

When you write out a Markov state space diagram or transition matrix, double-check that every row sums to 1, i.e. that the total probability of all edges out of a node is equal to 1.

Example 9.5 (Memory length).
In the Shakespeare example on page 98, the next character was chosen based on the preceding *two* characters, i.e. the causal diagram is



This is a real mess, hard to do calculations with. But there's a clever trick to turn it into a Markov chain: we simply 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]

def nextx(x):
    nextchar = random.choice(alphabet, next_chat_prob[x])
    return (x[-1], nextchar)

while True:
    prec = x[-1]
    newval = nextx(prec)
    x.append(newval)
```

This way of writing the code makes it clear that we choose the next element based only on the

previous element. After generating a sequence $(X_0, X_1, \dots)$ we can extract the actual characters $(C_0, C_1, \dots)$ as a post-processing step.

The rule "pick the next character 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.

* ✻ *

In IA *Discrete Mathematics* 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.

9.3. Calculations based on memorylessness

There are two big ideas to calculations with Markov chains:

- A Markov chain is memoryless, or in other words the causal diagram is

$$X_0 \to X_1 \to X_2 \to \cdots$$

  Memorylessness makes it very easy to do calculations, using the general strategies from section 9.1—namely (1) rewrite any probabilities to have the form 'probability of a child node, conditional on its parents', and (2) if the parents aren't already in the expression, find a way to incorporate them.

- The rule for generating the next state is the same at every timestep. This means that we can 'reset the clock' when we're doing probability calculations. For example, $\mathbb{P}(X_{13} = x \mid X_{10} = y)$ is equal to $\mathbb{P}(X_3 = x \mid X_0 = y)$.

Example 9.6 (Multi-step transition probabilities).
Consider the Markov model for Cambridge weather on page 103. If it's grey today, what's the chance it will be grey three days from today?

The question is asking us to calculate

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

Let's start by drawing the causal diagram. The underlying mechanism of a Markov chain is 'choose the next state based on the current state', so the causal diagram is

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

We're asked to calculate something about the distribution of X_3, and we're given X_0, so we'll use the general strategy "if the parents aren't already in the expression, find a way to incorporate them". We can do this using the law of total probability, conditional form (treating $\{X_0 = g\}$ as baggage):

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

The other general strategy is "rewrite any probabilities to have the form: probability of a child node, conditional on its parents". For the first term we get this from conditional independence:

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

For the second term, shift the random variables around using the definition of conditional probability (conditional form), to put the child node on the left:

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

Another application of conditional independence gives us what we want:

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

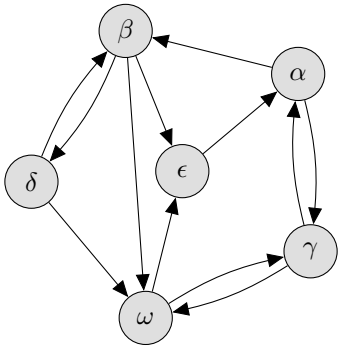
Now we can solve it. Rewriting it in terms of the transition matrix, it is

$$\begin{aligned} &= \sum_{x_1,x_2} P_{gx_1} P_{x_1x_2} P_{x_2g} \\ &= [P^3]_{gg} \quad \textit{when written in matrix form.} \end{aligned}$$

```
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
# returns the answer: 0.505
```

∎

Example 9.7 (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 $X = (X_n)_{n\geq 0}$. Let ξ be the quantity we want to calculate,

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

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 9.6. Instead, let's condition just on X_1. Using the law of total probability (conditional form),

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

The second part is just $P_{\alpha x_1}$. For the first part, the causal diagram tells us about conditional independence: conditional on X_1 the future is independent of X_0, thus

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

The final trick is to 'reset the clock'. It doesn't make any difference whether we start measuring time from $n = 0$ or from $n = 1$, thus

$$\mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 1\\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_1 = x_1\right) = \mathbb{P}\left(\begin{matrix}X \text{ hits } \omega \text{ at some } n \geq 0\\ \text{before hitting } \alpha\end{matrix} \;\middle|\; X_0 = x_0\right).$$

To streamline the notation, let's define π_x to be this hitting probability,

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

We've shown that ξ, the probability we want to calculate, is equal to $\sum_x P_{\alpha x}\pi_x$. We still need to find π_x. By repeating the entire conditioning argument we've just been through, it's easy to show

$$\pi_x = \sum_y P_{xy}\pi_y \text{ for } x \notin \{\alpha, \omega\}, \quad \text{and} \quad \pi_\alpha = 0\,,\ \pi_\omega = 1.$$

In Python we'll let π be a 6-dimensional vector with elements in $[0, 1]$, solve the equations with matrices, then extract $\sum_x P_{\alpha x}\pi_x = [P\pi]_\alpha$.

```
import numpy as np
# States are in the order [α,β,γ,δ,ϵ,ω]
# Set up an adjacency matrix for the graph, and scale it so rows sum to 1
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 Pπ = π i.e. (P − I)π = 0, except for π_α and π_ω
# Bundle all the equations together in a matrix, and solve with np.linalg.lstsq
```

```
A = P - np.eye(6)
A[0,:] = [1,0,0,0,0,0]
A[5,:] = [0,0,0,0,0,1]
b = np.zeros(6)
b[5] = 1
π = np.linalg.lstsq(A, b)[0] # [0, 0.333, 0.5, 0.667, 0, 1]

# Return the hitting probability we wanted to calculate
(P @ π)[0]  # 0.417
```

∎

There is a general theorem behind this example. You can find this and other results about hitting times etc. in standard textbooks on Markov chains[38]—though often it's just as much work to translate your problem into a theorem-ready version as it is to just solve it from first principles. Also, there will be many applications, especially in machine learning applied to Markov chains, where there aren't ready-made theorems.

Theorem (hitting probability) *. Let A be a subset of a Markov chain's state space. The hitting probability from x is

$$\mathbb{P}\big(\text{ever hit } A \,|\, \text{start at } x\big).$$

The hitting probabilities solve

$$\pi_x = \sum_y P_{xy}\pi_y \text{ for all } x \notin A\,, \quad \pi_x = 1 \text{ for all } x \in A\,.$$

Sometimes this system of equations has multiple solutions. In that case, the hitting probability from x is the minimum of π_x over all solutions $\pi \geq 0$.

For example 9.7, first delete the edges out of α and put a single link from α to itself. Let the original transition matrix be P, and call this modified matrix P'. Applying the theorem to P' with $A = \{\omega\}$, we get a solution $\pi'_x = \mathbb{P}(\text{ever hit } \omega \,|\, \text{start at } x)$. This is the same as the probability of hitting ω before α in the original Markov chain. The rest of the solution is as before.

∎

[38] J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997. URL: http://www.statslab.cam.ac.uk/~james/Markov/.

9.4. Stationary distribution

We'd like to be able to reason about the long-run average behaviour of a Markov chain.

One straightforward application is to systems modelling. We might for example use a Markov chain to model queueing behaviour at a supermarket (many queues, many servers), compared to queueing behaviour at the train station (one queue, many servers) to work out how the queueing regime affects average waiting time. Computer networking is full of systems modelling problems like this. Another interesting application is called Gibbs sampling, a clever way to speed up computational Bayesian inference. It will be taught in Part II *Machine Learning and Bayesian inference*.

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. Conversely, if π is a probability distribution such that

$$\mathbb{P}(X_0 = x) = \pi_x \text{ for all } x \qquad \Longrightarrow \qquad \mathbb{P}(X_n = x) = \pi_x \text{ for all } x \text{ and } n$$

then π is called a *stationary distribution* or *equilibrium distribution*.

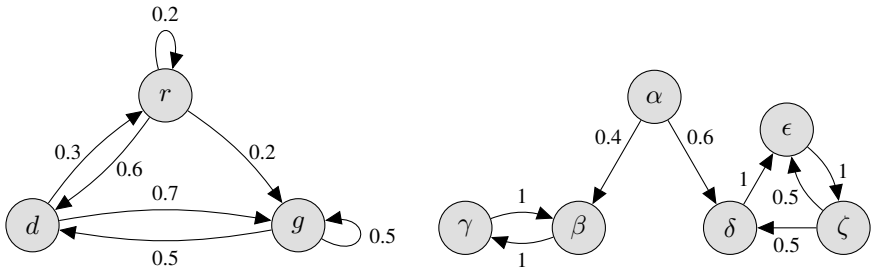
The word 'stationary' does not mean that the Markov chain has somehow stopped—a Markov chain is defined to go on forever, always stepping randomly from state to state. It is the *distribution* that is stationary i.e. unchanging.

Technically, all that stationarity tells us is 'if we pick the initial state X_0 randomly according to π, then X_1 will have distribution π, and X_2, and so on.' However, the stationary distribution turns out to be key for all sorts of other long-run average properties—it gives the long-run fraction of time spent in a given state; also, no matter where we start the chain, its distribution converges towards the stationary distribution. These further properties are described in the extended version of the notes.

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

$$\pi = \pi P, \quad \pi^\top 1 = 1. \tag{2}$$

We'll illustrate stationary distributions with two examples, the Markov chain for Cambridge weather from page 103, and a pathological case.



COMPUTING THE STATIONARY DISTRIBUTION

We can find a stationary distribution using the same sort of calculations based on memorylessness that we used in section 9.3. If X is a stationary Markov chain then

$$\mathbb{P}(X_n = x) = \sum_y \mathbb{P}(X_n = x \mid X_{n-1} = y)\ \mathbb{P}(X_{n-1} = y) \qquad \text{for all } x, n$$

hence a stationary distribution π must satisfy

$$\pi_x = \sum_y \pi_y P_{yx} \qquad \text{for all } x \tag{3}$$

where P is the transition matrix.

Exercise 9.8 (Computing the stationary distribution).
Find the stationary distribution of Cambridge weather.

Writing out in longhand the equations from (3)*,*

$$\begin{aligned}\pi_r &= 0.2\pi_r + 0.3\pi_d\\ \pi_d &= 0.6\pi_r + 0.5\pi_g\\ \pi_g &= 0.2\pi_r + 0.7\pi_d + 0.5\pi_g.\end{aligned}$$

Although there are three equations and three unknowns, when we try to solve them we find there is not a unique solution: if the vector π is a solution then so is $\kappa\pi$ for any constant κ. To pin π down we need an extra equation, an equation that comes from the fact that π is a probability distribution:

$$\sum_x \pi_x = 1.$$

Rather than solve all these simultaneous equations with algebra, we can turn them into matrix notation and then ask the computer to solve them. Equation (3) *becomes $\pi = \pi P$, or equivalently $(P-I)^\top \pi = 0$. The normalizing equation is $1^\top \pi = 1$. In Python,*

In numpy, if π is a one-dimensional array then it can be used either as a row vector or a column vector. In πP it is treated as a row vector, and in $(P-I)^\top \pi$ it is treated as a column vector.

```
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]
```

∎

Exercise 9.9 (Computing the stationary distributions).
Show that the pathological Markov chain has multiple stationary distributions. Find them all.

We can compute a stationary distribution for the pathological Markov chain using exactly the same method, but there is a problem: equation (3) *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 is

$$[\pi_\alpha, \pi_\beta, \pi_\gamma, \pi_\delta, \pi_\epsilon, \pi_\zeta] = a\left[0, 1/2, 1/2, 0, 0, 0\right] + (1-a)\left[0, 0, 0, 1/5, 2/5, 2/5\right] \tag{4}$$

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 (4) *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 $[1/5, 2/5, 2/5]$.*

∎

Example 9.10 (Irreducibility).
Are these two Markov chains irreducible?

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.

∎

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*.

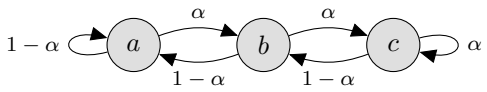
Theorem (detailed balance). Let X be a Markov chain with transition matrix P. If there is a vector π such that

$$\pi_x P_{x\,y} = \pi_y P_{y\,x} \quad \text{for all states } x \text{ and } y \tag{5}$$

then π solves $\pi = \pi P$. Equation (5) is called the *detailed balance* condition. This theorem is trivial to prove: just write out (3) and substitute in (5).

If the chain is irreducible, then the theorem on page 9.4 tells us that there is a unique stationary distribution. If we have found a distribution π that solves the detailed balance condition, then π must be that unique stationary distribution.

Exercise 9.11 (Stationary distribution using detailed balance).
Calculate the stationary distribution of the following Markov chain.



Is it irreducible? Actually, if $\alpha = 0$ or $\alpha = 1$ then the chain is not irreducible: if $\alpha = 0$ then it gets stuck in state a, so the stationary distribution is $\pi_a = 1$, $\pi_b = \pi_c = 0$. If $\alpha = 1$ then it gets stuck in state c, so the stationary distribution is $\pi_a = \pi_b = 0$, $\pi_c = 1$.

In the case $0 < \alpha < 1$, it's easy to see that it's possible to get from any state to any other. Therefore the uniqueness theorem of applies, i.e. 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 (2). *In this case, the detailed balance equations are*

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

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 9.12 (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 (2)*, 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*

$$\frac{m_x}{336} \times \frac{1}{m_x} = \frac{m_y}{336} \times \frac{1}{m_y} \quad \textit{if } x \leftrightarrow y \textit{ is legal,}$$
$$\frac{m_x}{336} \times 0 = \frac{m_y}{336} \times 0 \quad \textit{if } x \leftrightarrow y \textit{ is illegal.}$$

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 $\sum_x m_x/336 = 1$.

∎

The result described here can be generalized to a random walk on any undirected graph.