exists for each state i. Under a couple more conditions,[3] we have the stronger result,

$$\lim_{t \to \infty} P(X_t = i) = \pi_i \tag{6.5}$$

These quantities π_i are typically the focus of analyses of Markov chains.

The π_i are called **stationary probabilities**, because if the initial state X_0 is a random variable with that distribution, then all X_i will have that distribution.

6.3.1 Calculation of π

In Chapter 13 it is shown that the π_i are easy to find (in the case of finite state spaces, the subject of this section here), by solving the matrix equation

$$(I - P')\pi = 0 \tag{6.6}$$

subject to the constraint

$$\sum_i \pi_i = 1 \tag{6.7}$$

Here I is the $n \times n$ identity matrix (for a chain with n states), and $'$ denotes matrix transpose. (While these equations do hold for infinite state spaces, the material from here onward assumes a finite state space.)

6.3.1.1 Solving the Equations

Note that (6.7) can be written as

$$O'\pi = 1 \tag{6.8}$$

where O is a vector of n 1s.

[3] Basically, we need the chain to not be **periodic**. Consider a random walk, for instance: We start at position 0 on the number line, at time 0. The states are the integers. (So, this chain has an infinite state space.) At each time, we flip a coin to determine whether to move right (heads) or left (tails) 1 unit. A little thought shows that if we start at 0, the only times we can return to 0 are even-number times, i.e. $P(X_n = 0 \mid X_0 = 0)$ for all odd numbers n. This is a periodic chain. By the way, (6.4) turns out to be 0 for this chain.

Recall that one of the beginning motivations of linear algebra is the solution of simultaneous linear equations. (6.6) represents such a system, but the fact that we have the 0 vector on the right-hand side says that one of those n equations is redundant; the matrix $I - P'$ is not of full rank.

But in view of (6.8), let's replace the last row in the matrix $I - P'$ in (6.6), which is redundant anyway, by O', and correspondingly replace the last element of the right-side vector by 1. Now we have a nonzero vector on the right side, and a full-rank (i.e. invertible) matrix on the left side. This is the basis for the following code, which we will use for finding π.

```
findpi1 <- function(p) {
   n <- nrow(p)
   # find I-P'
   imp <- diag(n) - t(p)
   # replace the last row of I-P' as discussed
   imp[n,] <- rep(1,n)
   # replace the corresponding element of the
   # right side by (the scalar) 1
   rhs <- c(rep(0,n-1),1)
   # now use R's built-in solve()
   solve(imp,rhs)
}
```

6.3.1.2 Example: Mean Time Between Wins in Die Game

Consider the die game example above. Guess what! Applying the above code, all the π_i turn out to be 1/10. In retrospect, this should be obvious. If we were to draw the states 1 through 10 as a ring, with 1 following 10, it should be clear that all the states are completely symmetric.

6.3.1.3 Another Way to Find π

Here is another way to compute π, that **will also help illustrate some of the concepts.** Suppose (6.5) holds. Recall that P^m is the m-step transition matrix, so that for instance row 1 of that matrix is the set of probabilities of going from state 1 to the various states in m step. Putting that together with (6.5), we have that

$$\lim_{n \to \infty} P^n = \Pi \tag{6.9}$$

where the $n \times n$ matrix Π has each of its rows equal to π.

We can use this to find π. We take P to a large power m, and then each of the rows will approximate π. In fact, we can get an even better appoximation by averaging the rows.

Moreover, we can save a lot of computation by noting the following. Say we want the 16th power of P. We could set up a loop with 15 iterations, building up a product. But actually we can do it with just 4 iterations. We first square P, yielding P^2. But then we square *that*, yielding P^4. Square twice more, yielding P^8 and finally P^{16}. This is especially fast on a GPU (graphics processing unit).

```
# finds stationary probabilities of a Markov chain using matrix powers

altfindpi <- function(p,k) {
   niters <- ceiling(log2(k))
   prd <- p
   for (i in 1:niters) {
      prd <- prd %*% prd
   }
   colMeans(prd)
}
```

This approach has the advantage of being easy to parallelize, unlike matrix inversion.

6.4 Example: 3-Heads-in-a-Row Game

How about the following game? We keep tossing a coin until we get three consecutive heads. What is the expected value of the number of tosses we need?

We can model this as a Markov chain with states 0, 1, 2 and 3, where state i means that we have accumulated i consecutive heads so far. If we simply stop playing the game when we reach state 3, that state would be known as an **absorbing state**, one that we never leave.

We could proceed on this basis, but to keep things elementary, let's just model the game as being played repeatedly, as in the die game above. You'll see that that will still allow us to answer the original question. Note that now that we are taking that approach, it will suffice to have just three states, 0, 1 and 2; there is no state 3, because as soon as we win, we immediately start a new game, in state 0.

Clearly we have transition probabilities such as p_{01}, p_{12}, p_{10} and so on all equal to 1/2. Note from state 2 we can only go to state 0, so $p_{20} = 1$.

Here's the code below. Of course, since R subscripts start at 1 instead of 0, we must recode our states as 1, 2 an 3.