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.

### 6.4.1 Markov Analysis

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 and 3.

```r
p <- matrix(rep(0,9),nrow=3)
onehalf <- 1/2
p[1,1] <- onehalf
p[1,2] <- onehalf
p[1,3] <- onehalf
p[2,1] <- onehalf
p[2,2] <- onehalf
p[2,3] <- 1
findpi1(p)
```

It turns out that

$$
\pi = (0.5714286, 0.2857143, 0.1428571)
$$

(6.18)

So, in the long run, about 57.1% of our tosses will be done while in state 0, 28.6% while in state 1, and 14.3% in state 2.

Now, look at that latter figure. Of the tosses we do while in state 2, half will be heads, so half will be wins. In other words, about 0.071 of our tosses will be wins. And THAT figure answers our original question, through the following reasoning:

Think of, say, 10000 tosses. There will be about 710 wins sprinkled among those 10000 tosses. Thus the average number of tosses between wins will be about $10000/710 = 14.1$. In other words, the expected time until we get three consecutive heads is about 14.1 tosses.

### 6.5 A Modified Notebook Analysis

Our previous notebook analysis (and most of our future ones, other than for Markov chains), relied on imagining performing many independent replications of the same experiment.
6.6. SIMULATION OF MARKOV CHAINS

6.5.1 A Markov-Chain Notebook

Consider Table 2.3 for instance. There our experiment was to watch the network during epochs 1 and 2. So, on the first line of the notebook, we would watch the network during epochs 1 and 2 and record the result. On the second line, we watch a new, independent replication of the network during epochs 1 and 2, and record the results.

But instead of imagining a notebook recording infinitely many replications of the two epochs, we could imagine watching just one replication but over infinitely many epochs. We’d watch the network during epoch 1, epoch 2, epoch 3 and so on. Now one line of the notebook would record one epoch.

For general Markov chains, each line would record one time step. We would have columns of the notebook labeled \( n \) and \( X_n \). The reason this approach would be natural is (6.4). In that context, \( \pi_i \) would be the long-run proportion of notebook lines in which \( X_n = i \).

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

For instance, consider the 3-heads-in-a-row game. Then (6.18) says that about 57% of the notebook lines would have a 0 in the \( X_n \) column, with about 86% and 14% of the lines showing 1 and 2, respectively.

Moreover, suppose we also have a notebook column labeled \( \text{Win} \), with an entry Yes in a certain line meaning, yes, that coin flip resulted in a win, with a No entry meaning no win. Then the mean time until a win, which we found to be 14.1 above, would be described in notebook terms as the long-run average number of lines between Yes entries in the \( \text{Win} \) column.

6.6 Simulation of Markov Chains

Following up on Section 6.5, recall that our previous simulations have basically put into code form our notebook concept. Our simulations up until now have been based on the definition of probability, which we had way back in Section 2.3. Our simulation code modeled the notebook independent replications notion. We can do a similar thing now, based on the ideas in Section 6.5.

In a time series kind of situation such as Markov chains, since we are interested in long-run behavior in the sense of time, our simulation is based on (6.5). In other words, we simulate the evolution of \( X_n \) as \( n \) increases, and take long-run averages of whatever is of interest.

Here is simulation code for the example in Section 6.4 calculating the approximate value of the long-run time between wins (found to be about 14.1 by mathematical means above):
# simulation of 3-in-a-row coin toss game

def threeinrow(ntimesteps):
    consec = 0  # number of consec Hs
    nwins = 0   # number of wins
    wintimes = 0  # total of times to win
    startplay = 0  # time step 0
    for i in range(ntimesteps):
        if toss() == 'H':
            consec += 1
            if consec == 3:
                nwins += 1
                wintimes += i - startplay
                consec = 0
                startplay = i
        else:
            consec = 0
    wintimes /= nwins


toss = lambda: 'H' if runif() < 0.5 else 'T'

6.7 Example: ALOHA

Consider our old friend, the ALOHA network model. (You may wish to review the statement of the model in Section 2.5 before continuing.) The key point in that system is that it was “memoryless,” in that the probability of what happens at time k+1 depends only on the state of the system at time k.

For instance, consider what might happen at time 6 if $X_5 = 2$. Recall that the latter means that at the end of epoch 5, both of our two network nodes were active. The possibilities for $X_6$ are then

- $X_6$ will be 2 again, with probability $p^2 + (1-p)^2$
- $X_6$ will be 1, with probability $2p(1-p)$