Markov Chains

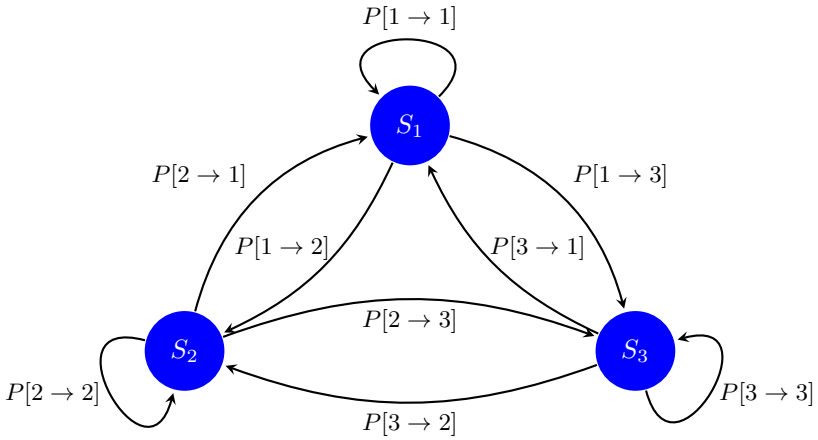
Prof. Wright

April 16, 2018

Markov Chain Review

A Markov chain on n states $S_1, S_2, \ldots, S_n$ is defined by a $n \times n$ transition matrix P, were the entry $P_{i,j}$ in row i, column j gives the probability $P[j \to i]$ of transitioning from state $S_j \to S_j$.

Here is a *state diagram* for three states.



The transition matrix of probabilities, with $p_{i,j} = P[j \to i]$, is

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

Under very general hypotheses, the transition matrix P, will have a steady state vector v_{ss} satisfying

$$Pv_{ss} = v_{ss}.$$

In other words, v_{ss} is an eigenvector of P corresponding to eigenvalue 1.

For example, define the following transition matrix of probabilities (note that each column sums to 1).

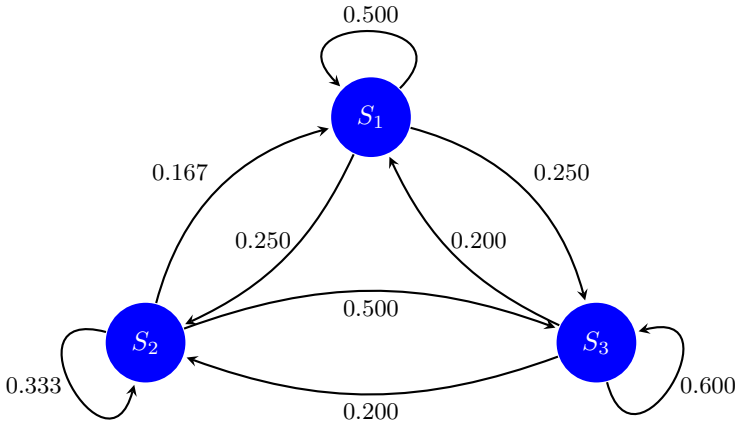
```
P = matrix(c(1/2,1/4,1/4,1/6,1/3,1/2,1/5,1/5,3/5),nrow=3)
colSums(P)
```

```
## [1] 1 1 1
```

```
P
```

```
##      [,1]      [,2] [,3]
## [1,] 0.50 0.1666667  0.2
## [2,] 0.25 0.3333333  0.2
## [3,] 0.25 0.5000000  0.6
```

Here is the corresponding state diagram:



Calculate the eigenstuff:

```
eigenStuff <- eigen(P)
eigenStuff
```

```
## eigen() decomposition
## $values
## [1] 1.0000000 0.2893150 0.1440184
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.4530270 -0.5927019 -0.1346523
## [2,] -0.4077243 -0.1899909 -0.6300988
## [3,] -0.7927972  0.7826928  0.7647511
```

The important information is that, yes, there is a eigenvalue of 1. It is also the largest eigenvalue. The corresponding eigenvector is the first one. We can access easily.

```
eigenVecs <- eigenStuff$vectors
steadyStateVec <- eigenVecs[,1]
steadyStateVec
```

```
## [1] -0.4530270 -0.4077243 -0.7927972
```

This isn't yet a steady state vector for the transition matrix. We need to **normalize** it so that the sum of the entries is 1.

```
steadyStateVec <- steadyStateVec/sum(steadyStateVec)
steadyStateVec
```

```
## [1] 0.2739726 0.2465753 0.4794521
```

This is the steady state distribution. The system will be in S_1 with probability 0.2739726, S_2 with probability 0.2465753, and S_3 with probability 0.4794521.

Agents again

Let's simulate this Markov chain using *agents*. We will use a large number of agents and watch how they all move around according to the probabilities defined by the transition matrix P. The idea is simple: if an agent is in state j, then it will move probabilistically to its next state according to the probabilities defined by $P[, j]$, the jth column of P.

Imagine a large number, N_1, of agents in state S_1. We can simulate where they go next using the `sample` function.

Suppose we have $N = 100$ agents in state S_1.

```
N1 <- 100
nextStates <- sample(1:3, N1, rep=T, prob=P[,1])
nextStates
```

```
##   [1] 2 2 2 3 1 1 3 2 1 3 3 1 1 3 2 2 1 3 2 3 1 1 1 1 3 1 1 1 1 1 2 1 3 1 1
##  [36] 3 1 2 3 1 1 2 2 2 2 1 2 3 3 1 2 1 1 1 3 1 1 3 2 2 3 1 2 3 1 1 1 3 3 3
##  [71] 1 1 1 3 1 3 1 2 3 1 1 3 3 3 3 1 1 1 2 1 1 3 1 1 3 3 1 1 1 1
```

Each of the 100 agents ended up in one of S_1, S_2 or S_3. Here's how to tally the totals using the `table` function. Doing so gives us the distribution of next states for these agents.

```
newDist <- as.vector(table(nextStates))
## do this to make sure all three states are represented.
newDist <- c(newDist, rep(0, 3-length(newDist)))
newDist
```

```
## [1] 50 20 30
```

In other words, out of our 100 agents,

- 50 ended up in S_1,
- 20 ended up in S_2,
- 30 ended up in S_3,

We could, of course, play the same game with agents in S_2 or S_3. Over time, all these agents move between the three states. Let's simulate this process.

Start with a fixed number of agents arbitrarily distributed between the three states:

```
numAgents <- 100000
agents = sample(1:3, numAgents, rep=T)
numEachState <- as.vector(table(agents))
numEachState
```

```
## [1] 33396 33511 33093
```

Hence, initially we have :

- 33396 agents in S_1
- 33511 agents in S_2
- 33093 agents in S_3

Now start them moving around:

```
nextState1 <- sample(1:3, numEachState[1], rep=T, prob=P[,1])
nextState2 <- sample(1:3, numEachState[2], rep=T, prob=P[,2])
nextState3 <- sample(1:3, numEachState[3], rep=T, prob=P[,3])
```

These aren't much to look at. Looking at the first 50 just shows where these agents ended up.

```
head(nextState1, 50)
```

```
##  [1] 3 3 1 3 2 3 1 2 2 1 3 3 1 1 3 3 3 3 3 1 3 1 2 1 2 1 2 2 1 2 3 1 1 1 1
## [36] 1 3 1 2 1 3 1 3 1 3 3 1 1 1 3
```

It helps to tally them as before.

```
fromState1 <- as.vector(table(nextState1))
fromState2 <- as.vector(table(nextState2))
fromState3 <- as.vector(table(nextState3))
```

Each of these represents the *flux* out of each state. For example:

```
fromState1
```

```
## [1] 16688  8304  8404
```

This means that

- 16688 agents went from S_1 to S_1
- 8304 agents went from S_1 to S_2
- 8404 agents went from S_1 to S_3

The other two `fromState` tables contain similar data.

Adding all these together gives the new distribution of the agents after one step of the Markov chain.

```
numEachState = fromState1 + fromState2 + fromState3
numEachState
```

```
## [1] 28876 26085 45039
```

Hence, after one step of the Markov chain, we have:

- 28876 agents in S_1
- 26085 agents in S_2
- 45039 agents in S_3

Compare this with what we had initially.

Of course, we want to do this a large number of times and see how the agents eventually distribute themselves across the states. We can do this easily:

```
## distribute agents uniformly
agents = sample(1:3, numAgents, rep=T)
numEachState <- as.vector(table(agents))
numEachState
```

```
## [1] 33248 33522 33230
```

```
numSteps <- 100
for(m in 1:numSteps){
  nextState1 <- sample(1:3, numEachState[1], rep=T, prob=P[,1])
  nextState2 <- sample(1:3, numEachState[2], rep=T, prob=P[,2])
  nextState3 <- sample(1:3, numEachState[3], rep=T, prob=P[,3])
  ##
  fromState1 <- as.vector(table(nextState1))
  fromState2 <- as.vector(table(nextState2))
  fromState3 <- as.vector(table(nextState3))
  ##
  numEachState <- fromState1 + fromState2 + fromState3
}
numEachState
```

```
## [1] 27629 24626 47745
```

It is better to look at these are proportions and compare to the steady state vector for the transitions matrix P.

```
numEachState/numAgents
```

```
## [1] 0.27629 0.24626 0.47745
```

```
steadyStateVec
```

```
## [1] 0.2739726 0.2465753 0.4794521
```

Pretty close! In other words, the agents are distributing themselves according the the steady state distribution probabilities. This is as it should be.

Look more closely at all the `fromState` vectors

```
fromState1
```

```
## [1] 13879  6814  6877
```

```
fromState2
```

```
## [1]  4137  8071 12338
```

```
fromState3
```

```
## [1]  9613  9741 28530
```

Notice that the fluxes between the states are nearly balanced—the number of agents leaving an state S_i nearly equals the number coming into S_i.

For example, at the end of the simulation, the number of states leaving S_1 is given by

```
fromState1[2] + fromState1[3]
```

```
## [1] 13691
```

In comparison, the number of agents coming into S_1 is

```
fromState2[1] + fromState3[1]
```

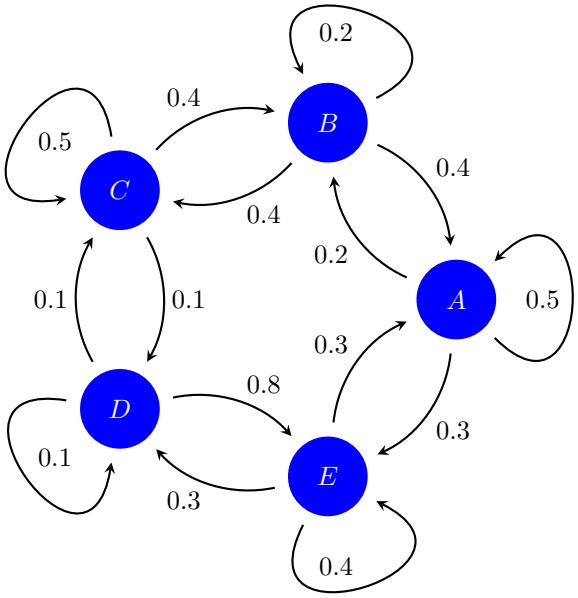
```
## [1] 13750
```

The numbers would be similar look at S_2 or S_3. This shows that the system is balanced, indicating that we have reached the steady state distribution of agents.

Exercises

1. Five-State Markov Chain

Consider the following state graph.



Build the 5×5 transition matrix for this state graph. Once you have it,

- Find the steady-state distribution.
- Repeat the agents simulation as above. That is, show that, starting with a reasonably large number of agents, that after a few hundred simulations, that the flow of agents in and out of each state are roughly balanced.

2. Simulating a Game

Construct a transtion matrix `TransProb` for a game that involves moving through 8 positions, arranged in a circle. These positions are the states of a Markov chain; we will denote the states as $S_1, S_2, \ldots, S_8$. S_1 in the first position, S_2 is the second position, and so on.

In the game, suppose that you roll a dice that tells you how many positions to move. After you reach S_8, the next position is S_1, and you go around again.

$$S1 \to S2 \to \cdots \to S8 \to S1 \to \cdots$$

Consider two versions of the game:

- **Version 1:** You move from one state to the next rolling a fair *3-sided die*. Hence if you are in S_3 and roll a 2, you move to S_5.
  If you are in S_7 and roll a 3 you move to S_2.
- **Version 2:** Same as Version 1, but if you land in S_5, then *proceed directly to* S_1.

Construct transition matrices for both games. For each, explore the steady-state behavior. In other words, what is the probability of being in a particular state, say S_3, after a large number of rolls? Is it different for the the different versions?

Do your work in this notebook, using RMarkdown to explain what you are doing and produce a nice-looking solution in HTML or PDF form.

3. Taxi Driver

A city has three different taxi zones, numbered 1, 2, and 3. A taxi driver operates in all three zones. The probability that the driver's next passenger has a destination in a particular one of these zones depends on where the passenger is picked up. Specifically, whenever the taxi driver is in zone 1, the probability the next passenger is going to zone 1 is .3, to zone 2 is .2, and to zone 3 is .5. Starting in zone 2, the probability that the next passenger is going to zone 1 is .1, to zone 2 is .8, and to zone 3 is .1. Finally, whenever the taxi is in zone 3, the probability the next passenger is going to zone 1 is .4, to zone 2 is .4, and to zone 3 is .2.

Use a Markov chain to simulate the taxi's location in the city, and answer the following questions:

- What are the long-term probabilities of the driver being in each zone of the city?
- If the driver starts the day in zone 3 and transports ten passengers before lunch, what is the probability that the driver will end up in zone 3 for lunch?
- Suppose that the fare for travel between zone 1 and zone 2 is $5, between zone 1 and zone 3 is $10, and between zone 2 and zone 3 is $8. On average, how much money would the taxi driver collect from transporting ten passengers? This may depend on where the driver starts. If so, in which zone should the driver start to maximize the expected revenue from the next ten passengers?