**Doodling with R: A simple Markov model**
Bodo Winter

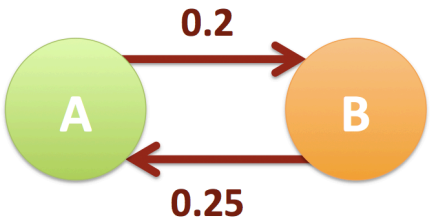
Let's have some fun with Markov processes and R! When I first learned about Markov processes, I was surprised by how simple they are… and by how easy they are to implement in any programming language.

So, quick background: What is a Markov process? A Markov process is characterized by…

(1) a finite set of states
(2) fixed transition probabilities between the states

Markov processes are all over the place in science. It's a very simple class of model, but it has applications in linguistics, biology, politics, sociology etc. Let's have a look at how these models work by going through a simple example.

Assume you have a classroom, with students who could be either in the state "alert" or in the state "bored"[1] (example taken from Scott Page's *Model Thinking* class). And then, at any given time point, there's a certain probability of an alert student becoming bored (say 0.2), and there's a probability of a bored student becoming alert (say 0.25). The following picture displays this very simple model of "classroom alertness".




So, say there are 20 alert and 80 bored students in a particular class. This is your initial condition at time point t. Given the transition probabilities above, what's the number of alert and bored students at the next point in time, t+1? Multiply 20 by 0.2 (=4) … and these will be the alert students that turn bored. And then multiply 80 by 0.25 (=20) … and these will be the bored students

[1] This example is taken from Scott Page's free *Model Thinking* class available online on coursera. That class is highly recommended!! Also, have a look at Scott Page's website: http://masi.cscs.lsa.umich.edu/~spage/

that turn alert. So, at t+1, there's going to be 20-4+20 alert students. And there's going to be 80+4-20 bored students. Before, 80% of the students were bored… and now, only 64% of the students are bored. Conversely, 36% are alert.

A handy way of representing this Markov process is by defining a transition probability matrix:

| | **A** | **B** |
|---|---|---|
| $\mathbf{A_{t+1}}$ | 0.8 | 0.25 |
| $\mathbf{B_{t+1}}$ | 0.2 | 0.75 |

What this matrix says is: A proportion of 0.8 of the people who are in state A (alert) will also be at state A at time point t+1. And, a proportion of 0.25 of the people who are in state B (bored) will switch to alert at t+1. This is what the first row says. The next row is simply one minus the probabilities of the first row, because probabilities (or proportions) have to add up to 1.

Now think about multiplying this matrix with the initial proportions of alert and bored students that we had above. 0.8 are bored and 0.2 are alert. In linear algebra this would look the following way:

$$\begin{bmatrix} 0.8 & 0.25 \\ 0.2 & 0.75 \end{bmatrix} \times \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix} \quad (1)$$

So, you multiply the transition matrix by a vector that contains the probabilities. In case you don't know how matrix multiplication works, the following spells out the calculations:

$$\begin{bmatrix} 0.8 \times 0.2 + 0.25 \times 0.8 \\ 0.2 \times 0.2 + 0.75 \times 0.8 \end{bmatrix} = \begin{bmatrix} 0.36 \\ 0.64 \end{bmatrix} \quad (2)$$

If you're not used to matrix multiplication, this might be a bit tricky. The best idea is to first compare (1) and (2) … to see which numbers are multiplied by which. Then, I recommend having a look at the Wikipedia article on matrix multiplication which (if you scroll down) has a nice visual image (http://en.wikipedia.org/wiki/Matrix_multiplication). If that wasn't enough, let me explain it to you in detail (you can skip this if you known matrix algebra well):

> Generally, you multiply rows in the first matrix with the corresponding columns of the second matrix. So here, you take the first element of the first row and multiply it by the first element of the vector, then you take the second element of the first row and multiply

it by the second element of the vector… and you add those two together. Then, you take the first element of the second row, multiplying it by the first element of the vector. Finally, you add to this the second element of the second row multiplied by the second element of the vector. The results of these calculations are exactly the proportions that we saw above: 36% alert student and 64% bored students.

This seems like a more complicated way of doing things than just multiplying the number of alert students by 0.2 and the number of bored students by 0.25 … but this matrix representation of the Markov process will come in very handy when we implement this model in R.

Now, you might ask yourself: What happens if this process continues? What happens at t+2, t+3 etc.? Will it be the case that at one point there are no bored students any more? Let's simulate this in R and find out!!

Open up R and your favorite script editor (e.g., Tinn-R for Windows, the built-in script editor for Mac). Start a new script. We first define a matrix. Let's call this "tmatrix" for "transition probability matrix":

```
tmatrix = as.matrix(c(0.8,0.25))
```

These numbers are the number of alert students who remain alert at t+1, and the number of bored students who turn alert at t+1. So this is supposed to be the first row of our transition probability matrix above. Type in "tmatrix" to see how this object looks like:

```
> tmatrix
     [,1]
[1,] 0.80
[2,] 0.25
```

O.k., this is not quite how we wanted it. To make this matrix look more like our matrix above, let's transpose it, using the function *t( )*. This function makes rows into columns and columns into rows.

```
tmatrix = t(tmatrix)
```

How does the matrix object look now?

```
> tmatrix
     [,1] [,2]
[1,]  0.8 0.25
```

Yeah, this is what we want. Remember that this row represents the proportion of alert students at t+1. Now let's add the second row, which will represent the proportion of *bored* students at t+1. For each column, we want the proportion of alert and bored students at t+1 to add up to 1. So, we can simply take 1 minus the first row and bind this to our existing matrix:

```
tmatrix = rbind(tmatrix,1-tmatrix[1,])
```

What does the matrix look like now?

```
> tmatrix
     [,1] [,2]
[1,]  0.8 0.25
[2,]  0.2 0.75
```

Awesome. Just how we want it. Let's give the rows and columns more meaningful labels.

```
colnames(tmatrix) = c("A","B")
rownames(tmatrix) = c("At+1","Bt+1")
```

```
> tmatrix
       A    B
At+1 0.8 0.25
Bt+1 0.2 0.75
```

This looks perfect! Just try to read this again and see whether you can make sense of it. 0.8 students who were in state A at time point t will still be in state A at t+1. And 0.25 students who were in state B at time point t will be in state A at t+1. The second row has a similar interpretation for alert and bored students becoming bored at t+1.

Remember that Markov processes assume fixed transition probabilities. This means that in the simulation that we'll be doing, we leave the transition probability matrix unchanged. However, we will define a vector of the actual proportions – and these are allowed to change. In time, we expect more and more students to become alert, because the transition probability from B to A (which, to remind you, was 0.25) is higher than from A to B (which was 0.2).

Let's set the initial condition… in the following step, I define a matrix called "smatrix" (short for "student matrix"), and I give it proportions 0.1 and 0.9. So, 0.1 students are alert and 0.9 students are bored. That's what we start out with in our little simulation:

```
smatrix = as.matrix(c(0.1,0.9))
rownames(smatrix) = c("A","B")
```

Looks good:

```
> smatrix
  [,1]
A  0.1
B  0.9
```

O.k., now we're ready to loop!

```
for(i in 1:10){
    smatrix = tmatrix %*% smatrix
    }
```

Here, we're looping 10 times and on each iteration, we multiply the matrix "tmatrix" with the student matrix "smatrix". We take this result and store it in "smatrix". This means that at the next iteration, our fixed "transition probability matrix" will be multiplied by a different student matrix, allowing for the proportions to slowly change over time.

Note that we used the operator "%*%". This is R's way of doing matrix multiplication, the calculation that we performed by hand above in matrix (2).

If you type in "smatrix", you'll see the outcome of our ten loop iterations:

```
> smatrix
          [,1]
At+1 0.5544017
Bt+1 0.4455983
```

So, after 10 iterations of the Markov process, we now have about 55% alert students and 45% bored ones. What is interesting to me is that even though 80% of the people who are alert at one time point remain alert at the next time point, the process only converged on 55% alert/45% bored after 10 iterations. What happens if you increase the number of iterations? Let's run a thousand iterations. When you do so, don't forget to re-initialize the matrix that represents the initial proportions of alert and bored students (0.1 and 0.9).

```
smatrix = as.matrix(c(0.1,0.9))
rownames(smatrix) = c("A","B")
for(i in 1:1000){
    smatrix = tmatrix %*% smatrix
    }
```

What's the outcome of this?

```
> smatrix
          [,1]
At+1 0.5555556
Bt+1 0.4444444
```

Interesting. A 1000 iterations, and we seem to be zoning in onto ~55% and ~44%. This phenomenon is called ***Markov convergence***. You could run even more iterations, and your outcome would get closer and closer to 0.5555 (to infinity)… So, the model converges on an equilibrium. However, this is not a fixed equilibrium. It's not the case that the Markov process comes to a hold or that nobody changes states between alertness and boredness any more. The equilibrium that we're dealing with here is a ***statistical equilibrium***, where the proportions of alert and bored students remain the same …. but there still is constant change (at each time step, 0.2 alert students become bored and 0.25 bored students become alert).

Markov models always converge to a statistical equilibrium if the conditions (1) and (2) above are met, and if you can get from any state within your Markov model to any other state (in the case of just two states, that clearly is the case).

What's so cool about this is that it is, at first sight, fairly counterintuitive. At least when I thought about the transition probabilities for the first time, I somehow expected all students to become alert… but as we saw, that's not the case.

Moreover, this process is not sensitive to initial conditions. That means that when you start with *any* proportion of alert or bored students (even extreme ones such as 0.0001 alert students), the process will reach the statistical equilibrium – albeit sometimes a little faster or slower. You can play around with different values for the "smatrix" object to explore this property of Markov convergence. Another interesting thing is that the process is impervious to intervention: Say, you introduced something that made more students alert – the Markov model would quickly get back to equilibrium.

So Markov processes are essentially *ahistorical* processes: history doesn't matter. Even with extreme initial conditions or extreme interventions, the process quickly converges to the equilibrium defined by the transition probabilities. The only way to persistently change the system is to change the transition probabilities.

Finally, what I find so cool about Markov processes is their computational simplicity. This was all the code we needed for our little demonstration:

```
tmatrix = as.matrix(c(0.8,0.25))
tmatrix = t(tmatrix)
tmatrix = rbind(tmatrix,1-tmatrix[1,])
colnames(tmatrix) = c("A","B")
rownames(tmatrix) = c("At+1","Bt+1")

smatrix = as.matrix(c(0.1,0.9))
rownames(smatrix) = c("A","B")
for(i in 1:1000){
    smatrix = tmatrix %*% smatrix
    }
```

… and I think we got a lot of conceptually interesting results from just a little bit of code.

Awesomeness!