Representing Markov Chains in LaTeX

Aditya Sengupta

September 14, 2020

1 Introduction

Markov chains in LaTeX look nice!

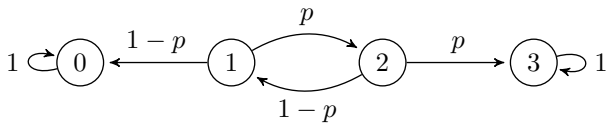

Figure 1: The "Gambler's Ruin" Markov chain

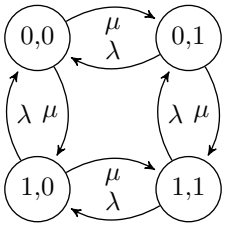

Figure 2: A continuous-time Markov chain representing two switches

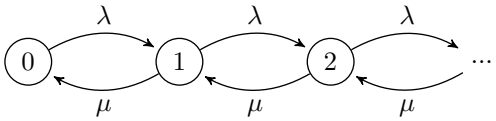

Figure 3: A continuous-time birth-death Markov chain

However, writing them can be difficult. LaTeX is very customizable, and there are usually multiple ways to reach the same output. This document aims to show some of the simplest ways of representing Markov chains.

2 Setup

You'll need the package `tikzpicture` and the Tikz library for shapes, arrows, and positioning. Add the following to your preamble (the part before `\begin{document}`):

```
\usepackage{tikz}
\usetikzlibrary{shapes,arrows,positioning}
```

All of the scripts to generate Markov chains should go inside `tikzpicture` environments, and the additional libraries let us set parameters that make the chains look nicer. Ideally, they should also be enclosed in `figure` environments and have a label and caption, as follows.

```
\begin{figure}
    \centering
    \begin{tikzpicture}[->,>=stealth',shorten >=2pt, line width=0.5pt,
       node distance=2cm]
    ...
    \end{tikzpicture}
    \caption{Your caption here}
    \label{fig:your_label_here}
\end{figure}
```

For what follows, it is assumed that all code shown is within a `tikzpicture` environment.

3 Placing nodes and relative positioning

Tikz allows you to set absolute coordinates, but it's usually tougher to keep track of these than it is to put down one position and then to reference later ones relative to the first one. The `positioning` library that we imported above gives us tools to do this, so that ideally, we'll never have to specify an (x, y) position for a node or arrow.

Let's start by placing a node with value 0. The syntax to do this is:

```
\node [circle, draw] (zero) {0};
```

`\node` is the general way we refer to any object whose location is important. In this line, we're specifying that we want to `draw` a `circle`, name it `zero`, and give it the label 0. The name `zero` is how we'll refer to this node later. Let's look at what this line does!

Figure 4: A zero node

Figure 4 shows us this zero node.

Now, what if we wanted to add a second node? Suppose we want to put down 1. We might write this:

```
\node [circle, draw] (zero) {0};
\node [circle, draw] (one) {1};
```

And we'd see the output in Figure 5.

Figure 5: A zero node and a one node, with no position specified

Oops. Looks like we need to do a bit more. As it is, tikzpicture just positioned both of them at the center of the figure, without looking for where `zero` was when it put down `one`. We'd like to be able to tell it to place `one` to the right of `zero`, and luckily, there's a way to do that!

```
\node [circle, draw] (zero) {0};
\node [circle, draw] (one) [right of=zero] {1};
```

Now, we've told it to put `one` to the right of `zero`. Figure 6 shows us what that looks like!

Figure 6: A zero node, and a one node to its right

This syntax extends to other relative positions: we could also have put `one` to the `left of`, `below of`, or `above of=zero`.

Note that the spacing between the two nodes was given by the `node distance` parameter from `\begin{tikzpicture}`. If we wanted to, we could change this. In Figure 7, it's 1cm instead of 2cm.

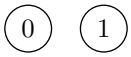

Figure 7: A zero node, and a one node closer to its right

4 Absolute positioning

Most of the time, relative positioning will work fine, but it's worth knowing how absolute positioning works too. Instead of specifying a distance from a previous node, it's possible to just use (x, y) coordinates and describe everything explicitly that way. To do this, we specify the coordinates after the label name by saying `(name) at (x, y)`. For example, if we did Figure 6 with absolute coordinates, the syntax might look like this:

```
\node [circle, draw] (zero) at (0, 0) {0};
\node [circle, draw] (one) at (2, 0) {1};
```

And you can see in Figure 8 that the result matches what we had before.

Figure 8: A zero node, and a one node to its right, using absolute coordinates

Most of the time, you probably won't need absolute coordinates, but they can be useful sometimes. For example, Figure 9 shows the most intricate Markov chain I've TeXed (with labels redacted in case you see this problem!) In this case I found it easiest to use absolute positioning: six of the seven nodes are on the same circle, so I just picked a convenient radius value r and set their positions to $(r\cos\theta, r\sin\theta)$ for θ increasing in 60-degree steps.

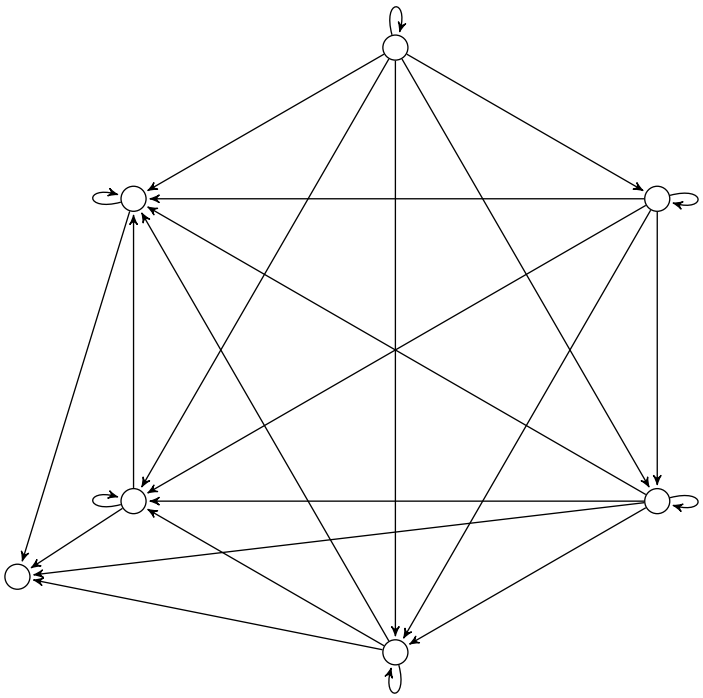

Figure 9: The overly complicated Markov chain

5 Arrows between nodes

Positioning the nodes is only half of drawing Markov chains: you also have to depict the connections between them. The easiest way to do this is with straight lines. Let's put a straight line between our zero and one nodes!

```
\node [circle, draw] (zero) {0};
\node [circle, draw] (one) [right of=zero] {1};
\path (zero) edge (one);
```

And we see the results in Figure 10.

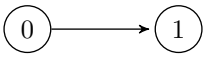

Figure 10: Zero going to one

Putting a label on top of that is also easy, using another `node`; this results in Figure 11.

```
\node [circle, draw] (zero) {0};
\node [circle, draw] (one) [right of=zero] {1};
\path (zero) edge node [above] {$1$} (one);
```

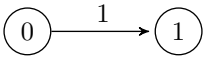

Figure 11: Zero going to one, with a label

6 Curved arrows between nodes

Another tool we'll need is curved arrows between nodes. This is useful to make sure that if you have two nodes, and both of them have a path to the other one, they don't overlap: instead, you can have one path curve upwards and one downwards.

To do this, we just have to require that an edge `bend left` or `bend right`, as the case may be. For example, let's have the zero-to-one Markov chain bend above (to the `left` of the arrow as it's leaving) and below (to the `right`)!

```
\node [circle, draw] (zero) {0};
\node [circle, draw] (one) [right of=zero] {1};
\path (zero) edge [bend left] node [above] {$1$} (one);
```

Figure 12 shows the two versions of this: one as it is in the code block above, and one with 'left' changed to 'right'.

Figure 12: Bend above and below

We can use this to make a loop, as in Figure 13!

```
\node [circle, draw] (zero) {0};
\node [circle, draw] (one) [right of=zero] {1};
\path (zero) edge [bend left] node [above] {$1$} (one);
\path (one) edge [bend left] node [below] {$1$} (zero);
```

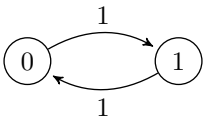

Figure 13: "ping pong"

7 Self-loops

Finally, we'll need to be able to make self-loops: this is done by changing the keyword `bend`, from the previous section, to `loop`. Loops still go left or right, but they can also go `above` or `below`: Figure 14 shows all these options.

```
\node [circle, draw] (zero) {0};
\path (zero) edge [loop below] (zero);
\path (zero) edge [loop above] (zero);
\path (zero) edge [loop left] (zero);
\path (zero) edge [loop right] (zero);
```

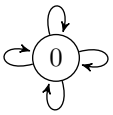

Figure 14: Zero with all four possible self-loops

8 Putting it all together

Let's take a look at the code that generated Figure 1 from the very start – you should be able to recognize all the elements we've talked about!

```
\node [circle, draw] (zero) {0};
\node [circle, draw] (one) [right of=zero] {1};
\node [circle, draw] (two) [right of=one] {2};
\node [circle, draw] (three) [right of=two] {3};
\path  (zero) edge [loop left] node {$1$} (zero);
\path (one) edge node[above] {$1 - p$} (zero);
\path (one) edge[bend left] node[above]{$p$} (two);
\path (two) edge node[above]{$p$} (three);
\path (two) edge[bend left] node[below]{$1 - p$} (one);
\path  (three) edge [loop right] node{$1$} (three);
```

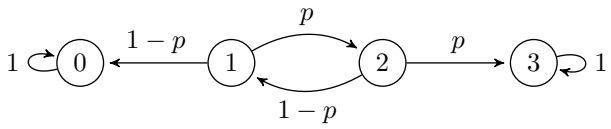

Figure 15: The "Gambler's Ruin" Markov chain, again (Figure 1 repeated)

References

[1] "TikZ and PGF", https://www.bu.edu/math/files/2013/08/tikzpgfmanual.pdf

[2] A lot of Stack Overflow trawling