Probability distribution of Markov Chains

Toshiki Arita

December 13, 2014

1 My reason for picking this topic

I was browsing some coding exercises under graph theory and I stumbled upon one for Snakes and Ladders. It basically gave a short overview of how the game can be completely modeled by a Markov chain due to the rules of the game. I decided to look up some more information about it because I've heard of Markov chains but never looked into it. Eventually I came across a blog post on R-bloggers (http://www.r-bloggers.com/basics-on-markov-chain-for-parents/) by Arthur Charpentier about the game and how it can be used as a tool by parents to model Markov Chains.

> "Markov chains is a very interesting and powerful tool. Especially for parents. Because if you think about it quickly, most of the games our kids are playing at are Markovian. For instance, snakes and ladders..."

I noticed that he used the matrix power formula : $x^n = x^0 P^n$ But there is a second method to solving it : $x^n = x^{(n-1)} P$ which is what I used.

2 What is probability distribution?

Probability distribution considers all the possibilities of an experiment and decides the likelihood of each individual outcome occuring.

3 What is a Markov Chain?

A discrete-time Markov chain is a system that goes through transitions disregarding prior states. Each transition is decided randomly and is described as memoryless because the next state only depends on the current. This

memoryless property is called the Markov property.

Markov chains are used in many statistical models of real world processes. Popular examples include : board games played with dice, predicting the weather, and the drunkard's walk.

4 What is Snakes and Ladders?

Snakes and Ladders, also called Chutes and Ladders, is a classic boardgame originating from ancient India. It is often represented as a 10x10 square numbered 1-100. Each player takes a turn to roll a dice, with 1 or more sides, allowing them to move through the grid. Throughout the board there are ladders which when landed on, allow the player to move up the grid a certain amount of spaces. There are also snakes which move the player back a certain amount of spaces. The original goal is to land exactly on 100 but in this paper, any roll which lands on or goes past 100 will be accepted.

5 Markov plays Snakes and Ladders

The game of Snakes and Ladders can be represented exactly as an absorbing Markob chain where the absorption state is the win condition. The chain is represented by a transition matrix of size 101x101. 101 is used because the extra state allows us to accomodate the player's off board position. Each position n represented as rows, has a set probability to transition to cells n+1...n+dice sides which is represented as columns. For example, assume a board of size 10, represented by a 11x11 matrix, with 3 dice sides. The transition matrix would be represented as

	0	1	2	3	4	5	6	7	8	9	10
0	0.00	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	0.00	0.00	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.33	0.33	0.33	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.33	0.33	0.33	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.33	0.33	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.33	0.33	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.33	0.33
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.67
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

```
10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
```

The matrix will change depending on ladders and snakes used. For example if there was a ladder from 5 to 6 and snakes from 4 to 3, 8 to 7 then the matrix would be represented as

```
   0    1    2    3    4    5    6    7    8    9    10
0  0.00 0.33 0.33 0.33 0.00 0.00 0.00 0.00 0.00 0.00 0.00
1  0.00 0.00 0.33 0.67 0.00 0.00 0.00 0.00 0.00 0.00 0.00
2  0.00 0.00 0.00 0.67 0.00 0.00 0.33 0.00 0.00 0.00 0.00
3  0.00 0.00 0.00 0.33 0.00 0.00 0.67 0.00 0.00 0.00 0.00
4  0.00 0.00 0.00 0.00 0.00 0.00 0.67 0.33 0.00 0.00 0.00
5  0.00 0.00 0.00 0.00 0.00 0.00 0.33 0.67 0.00 0.00 0.00
6  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.67 0.00 0.33 0.00
7  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.33 0.00 0.33 0.33
8  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.33 0.67
9  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
```

6 The algorithm

We can determine the probabilities of each cell after n rolls by calculating
$x^n = x^{(n-1)}P$
Where x is the probability after n events and P is the transition matrix. Each row corresponds with the nth event.

In R

```
find_prob_dist<-function(initial,trans_mat,board_size){
  prob_dist<-matrix(0.0,nrow=board_size+1,ncol=board_size+1)
  rownames(prob_dist) = 0:board_size
  colnames(prob_dist) = 0:board_size

  # Simulate experiments
  for(count in 1:(board_size+1)){
    if(count == 1){
      prob_dist[1,] <- as.matrix(col) %*% trans_mat
```

```
    }
    else{
      temp <- prob_dist[(count-1),] %*% trans_mat
      prob_dist[count,] = temp
    }
  }
  return (prob_dist)
}
```

In C++

```
NumericMatrix  simulate_rolls_cpp(NumericMatrix init, NumericMatrix trans_mat) {
  NumericMatrix temp;
  NumericMatrix output(trans_mat.nrow(), trans_mat.ncol());
  NumericMatrix temp_vec(1,trans_mat.ncol());
  //    simulate_rolls(col,trans_mat)

  for (int i = 0; i < trans_mat.nrow(); i++) {
    if (i == 0) {
      temp = mat_mult_cpp(init, trans_mat);
      output(0,_) = temp(0,_);
    }
    else {
      temp_vec(0,_) = output((i-1),_);
      temp = mat_mult_cpp(temp_vec, trans_mat);
      output(i,_) = temp(0,_);
    }
  }


  return output;
}
```

Would both return (assuming precision is set to 2 decimals)

	0	1	2	3	4	5	6	7	8	9	10
1	0.00	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.11	0.56	0.00	0.00	0.33	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.26	0.00	0.00	0.41	0.22	0.00	0.11	0.00
4	0.00	0.00	0.00	0.09	0.00	0.00	0.17	0.35	0.00	0.21	0.19
5	0.00	0.00	0.00	0.03	0.00	0.00	0.06	0.23	0.00	0.17	0.51

```
6  0.00 0.00 0.00 0.01 0.00 0.00 0.02 0.12 0.00 0.10 0.76
7  0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.05 0.00 0.04 0.89
8  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.02 0.00 0.02 0.96
9  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.00 0.01 0.98
10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.99
11 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
```

However, the running speed is hugely different even though the algorithm is similar.

```
Unit: microseconds
                   expr       min        lq    mean  median      uq  max neval
find_prob_dist     92.170 95.5280 110.36282 97.7665 100.335 927.627        100
find_prob_dist_cpp 8.084  8.9285  10.93835  10.1710  12.245  27.573        100
```