Simulating dice game

Jingyi Cui

Due: 5/8

Contents

Introduction

In this problem, we will discover things about rolling a die. Suppose we are playing a dice game. We begin with a score of zero. Then, we roll a die. If the face value that shows up divides the sum[1], then the face value will be added to the sum. If the face value does not divide the sum, the face value will be subtracted from the sum. Notice that the sum will be nonnegative; thus, if the sum after being the subtraction, it is replaced with zero. We continue rolling the die until the sum is 20 or more.

This problem is an example of Markov Chain, since it is a system that probability of transition into a state depends only on the current state(current sum) and is unaffected by the states of the system at earlier times. This system can change its state only at discrete set of times: each time we roll the die. In this chain, there are twenty-one states, from 0 to 20; each state indicates our current sum. State 20 is the one absorbing state. At each step, the chain is in exactly one of twenty-one states. And we start this game in state 0 with a sum of zero. Below is the code for generating the transition matrix A and matrix A. It tells us the probability from one state to another. Each entry a_{ij} of A gives the probability of transitioning from state i to state j in one step.

```
A <- matrix(0, nrow = 21, ncol = 21)
count = 0
for (i in 0:14) {
  for (j in 1:6) {
    if(i%%j == 0){
      A[i + 1, i + j + 1] = 1/6
    }else{
      if(i - j <0){
        A[i+1, 1] = A[i+1, 1] + 1/6
      }else{
        A[i+1, i - j+1] = 1/6
      }
    }
  }
}
for(i in 15:20){
  for(j in 1:6){
    if(i + j >= 20){
      if(i%%j == 0){
        A[i+1, 21] = A[i+1, 21] + 1/6
      }else{
        A[i, i - j] = 1/6
      }
    }else{
      if(i%%j == 0){
        A[i, i + j] == 1/6
      }else{
        A[i, i - j] = 1/6
      }
    }
  }
}
A[21, 21] = 1
```

[1]We say A divides B if there exists an integer k such that $B = Ak$

$$A = \frac{1}{6}\begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6
\end{pmatrix}$$

We know that A is the transition matrix for a Markov chain and A^n_{ij} gives the probability that the chain will be in state j exactly n steps after being in the state i.

I use R to simulate this game and here is my code:

```
#set.seed(123)
numberOfGames = 100000
bigSum = 0

for (i in 1:numberOfGames) {
  sum = 0
  rollCount = 0
  while(sum < 20){
    rollCount <- rollCount + 1
    roll <- floor(runif(1, min = 1, max = 6))
    # print(roll)
    if(sum%%roll == 0){
      sum = roll + sum
    }else{
      sum = sum - roll
    }
    if(sum < 0 ){
      sum = 0
    }
  }
  bigSum = bigSum + rollCount
}
print(bigSum/numberOfGames)
```

Since I did not set the seed, the answer will be different each time. I run this code for five times and the output value, which is number of rolls on average, is 136.923, 136.7995, 136.4966, 136.5724 and 136.5758.

When I raise A to the 10th power, I notice that e$A[2,1]$ is 1481520427572973/7312316880125952. This means that if the chain starts in state 1, then the probability that it is in state 0 after 10 steps is approximately 1.935%. In addition, following the theorem, I calculate $N = (1-Q)^{-}1$, where Q is the non-negative matrix that arise from the transition probabilities between non-absorbing states(in this example, absorbing states is 20, since we never leave once entered). And I sum up the first row yielding the expected number of steps until we are in absorbing state 20. The answer is 79585090538805/689945209271 which can be round to 115 rolls approximately. So on average, it will take about 115.35 tolls until the um is 20 or greater. This is kind of different from the answer I get in the simulation. I guess maybe the way I choose to generate the random number is different. I just use the function $runif$ to generate a number between 1 and 6 instead of generating a number between 0 and 1 and multiple by 6. In addition, the way of generating random number from R is using the uniform distribution. I guess all these will effect the final answer: roll counts in average.

The central limit theorem says that these estimates should be close to normal distributed. If we just use the fact that normally distributed implies symmetric about the mean, then we can estimate the probability that fewer than 50 rolls. Since we already have the expected roll count which is around 136, we can estimate using the exponential distribution with $\lambda = 136$. Suppose x i the number of rolls to get the sum of 20 and $x \sim exp(\lambda)$, $\lambda = \frac{1}{136}$. Thus, $P(x < 50) = 1 - e^{-50\lambda} = 0.308$, $P(x < 100) = 1 - e^{-100\lambda} = 0.521$ and $P(x < 200) = 1 - e^{-200\lambda} = 0.770$. For calculating number of rolls making the probability of sum being 20 or more, suppose n is the roll count and $P(x < n) = 1 - e^{\frac{-n}{136}} = 0.9$. The answer I get is 313 rolls. And in this problem, there is no upper bound on the number of rolls needed to reach a sum of at least 8.