Bob Lin
Math381
Homework5

The problem is to roll a die (fair and six-sided). We will add or subtract number to get a score. Here are the rules:

1. When the number we roll divides the current score, we update score by **adding** that number to our current score.
2. When the number we roll **does not** divide the current score, we update score by **subtracting** that number to our current score.
3. If the score becomes negative, we set it to zero.
4. The game starts with 0 score and stop when we get 20 or more score .

Then, I will use Markov chain to express all possibility of transition:

Define score S = 0 to be the initial state, and $S \geq 20$ to be goal state. Each row in that matrix will be a different current state from 1 to 21, and each the column will give me the probability to arrive another state. In the matrix, A_{ij} is the probability we can get to state j from state i in one step, and the last state is our absorbing state.

I generate six times the transition matrix by python code and adapting the rules above:

```
for row in range(0,20):
    output = ["0"]*21
    sum21 = 0
    sum1 = 0
    for col in range(row + 1,row + 7):
        if (row)%(col - row) == 0:
            if col < 20:
                output[col] = "1"
            else:
                sum21 = sum21 + 1
        else:
            sub = 2*row - col;
            if sub > 0:
                output[sub] = "1"
            else:
                sum1 = sum1 + 1
    output[0] = str(sum1)
    output[20] = str(sum21)
    print(output)
absorb = ["0"] * 21
absorb[20] = "1"
print(absorb)
```

They create following matrix, and divided by 6 will be our transition matrix:

[0, 1, 1, 1, 1, 1, 1, 0, 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, 0]
[4, 0, 0, 1, 1, 0, 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, 0]
[2, 1, 0, 0, 0, 1, 1, 0, 1, 0, 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]
[0, 1, 1, 0, 0, 0, 0, 1, 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, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 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, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 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, 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, 1]

With this matrix, I will further explore several properties about the game:

<u>First:</u>

I want to see the expected number of rolls for me to make the score at least 20.

The matrix transition A can be written in canonical form, which is:

$$A = \begin{pmatrix} Q & R \\ O & J \end{pmatrix}$$

Where Q is the matrix without the last row and last column of matrix A, J is the identity matrix with 1 in diagonal and 0 in elsewhere, and O is a matrix with all zeros.

By using the Q above, a theorem shows that $N = (I - Q)^{-1}$ gives important information of:

The sum of the i-th row of N gives the mean number of steps until absorbtion when the chain is started in state i.

So, The sum of first row of N is the expected number of steps until we are in absorbing state 20, then by using Matlab code:

```
Q = zeros(20)
```

```
for row = 1:20
    for col = 1:20
        Q(row,col) = A(row,col)
    end
end
N = inv(eye(20) - Q);
sum = 0;
for col = 1:20
    sum = sum + N(1,col)
end
format long
sum
```

we get that the sum of first row of N equals to 115.3498704960856, so we need about 115.35 rolls on average in order to get a score equal or greater than 20. Here, there may be a rounding error, because 0.3333333 is not equals to 1/3, and the rounding error is about 0.00000003333…. In this question, because each number in first row will create a rounding error less than 0.0000000000001, so 21 numbers will totally create a rounding error less than 0.0000000000021

Second:

I want to see the probability for me to get a score at least 20 under different times limits:

Here, following from the definition of the transition matrix, if A_{ij} is the transition matrix of the possibility we can get to state j from state i in one step, A_{ij}^{k} is the probability that we can get from state i to state j in k steps. Thus, by setting i to 1, j to 21, we can get the probability to the goal state in k or fewer rolls, so by simply setting k to 50, 100, and 200, The last item in first row is the probability we want.
This can be computed by following Matlab code:

```
k = 50; % can switch to 100 and 200
Achange = A^k;
prob = Achange(1,21);
```

Then, I got following data:

Times -	≤ 50	≤ 100	≤ 200
Probability	0.33871716250075	0.579198962236171	0.829605370069131

Here, it's the same idea as first question, but this time, because we do the matrix multiplication. During each time, we multiple two number to get a new one, so if first number is (exact1 – error), and second number is (exact2 – error2), when they times together, the rounding error of new number is

$$error1\ error2 - error2\ exact1 - error1\ exact2$$

So after doing matrix multiplication 50 times, the error will be obvious.

Third:

I want to see how many rolls will guarantee that there is at least 90% for me to get a score at least 20.

In last question, if I roll 200 times, there is only 83% for me to get to the goal state, so I need more to guarantee the at least 90%, so I explore from 200th times until I get the probability equals or greater than 90 with following Matlab code:

```
k = 200; % can switch to 50 and 200
Achange = A^k;
prob = Achange(1,21);
while prob < 0.9
    k = k + 1
    Achange = A^k
    prob = Achange(1,21);
end
```

Then, the result of k is 259 with probability 0.900043837709102, which means that we need 259 rolls to guarantee that there is at least 90% for me to end in goal state.

To better understand the game, I will simulate it by using a random die in code:

```
import random

n = 10;
dice = [1, 2, 3, 4, 5, 6]
def roll():
    number_rolls = 0;
    sum = 0;
    while sum < 20:
        rolls = random.randint(1,6)
        if sum % rolls == 0:
            sum = sum + rolls
        else:
            sum = sum - rolls
            if sum < 0:
                sum = 0
        number_rolls = number_rolls + 1
        '''print(str(number_rolls), "roll is:", str(rolls), "sum is:",
str(sum))'''
```

```
    return number_rolls
def main():
    total_sum = 0
    for i in range(0, n):
        itr = roll()
        total_sum = total_sum + itr
        print(i + 1, "roll times is:", str(itr))
    print("average times to goal state is", total_sum / n)

main()
```

By changing the value of n, I can see the average times needed to get to goal state, and I try different n and get following result:

N (run time)	Average times
10 (1s)	169.3
100 (1s)	106.62
1000 (1s)	114.685
10000 (1s)	114.9773
100000 (20s)	115.39164
1000000 (around 3min)	115.417307

These data show that when n increases to a very large number, our average times will become stable and approximate to the expected number which is 115.35.