Math 381 A
Sheng Yu

Homework #5 Writeup: Dice Game

I. Introduction:

For a particular dice game, we consider prime numbers are bad. We start with a fair, six-sided die with 1, 2, 3, 4, 5, 6 on each side, and our score starts with zero. Then we roll the die, if our score plus the roll is not a prime number, then the result becomes our new score. But if the result ends up being a prime number, there are two cases in here. If our current score is zero, then we will take whatever the roll we get as our new score no matter it is a prime number or not. If our current score is not zero, then we will subtract the roll from our current score, and if our score ever becomes less than zero, replace it with zero. There will be a limit, and the game ends when the score reaches or exceeds the limit. We will use the Markov chains to investigate this game by generating the transition matrix and then use the matrix to study the game to get some useful information.

II. Markov Chains:

We start with the limit being 26. With Markov Chain, we can model the game with 27 states: 0, 1, 2, 3, 4, 5, …, 26. State 26 is the one absorbing state which means if our score reaches 26 or higher, the game will end. Let A be the transition matrix for this chain, then A would be a 27 by 27 matrix.

With the game rules introduced in previous page, we can generate the matrix with the following MATLAB code:

```
function transMatrix = transMatrix(limit)

n = limit + 1;      % Since we always include 0 as one of the states.
A = zeros(n, n);
A(n, n) = 1;
p = 1/6;            % Probability of getting one of the six sides of a dice.

for i = 1:limit
    for j = 1:6
        % Refresh the score for all six possibilities
        score = i - 1;
        % Case the sum is prime number
        if isprime(score + j)
            if score + j >= limit
                A(i, n) = A(i, n) + p;
            elseif score == 0
                score = score + j;
                A(i, score + 1) = A(i, score + 1) + p;
            else
                score = score - j;
                if score <= 0
                    A(i, 1) = A(i, 1) + p;
                else
                    A(i, score + 1) = A(i, score + 1) + p;
                end
            end
        % Case the sum is not prime number
        else
            score = score + j;
            if score >= limit
                A(i, n) = A(i, n) + p;
            else
                A(i, score + 1) = A(i, score + 1) + p;
            end
        end
    end
end

transMatrix = A;

end
```

The code above is a MATLAB function that takes the limit of our game as parameter, and it returns the transition matrix as output, we can use it as this:

```
% Generate the transition matrix

limit = 26;

A = transMatrix(limit);
```

We can then generate matrix Q, and N, and the expected number:

```
Q = A(1:limit, 1:limit);
N = inv(eye(limit) - Q);
expected_nun = sum(N(1, :));
```

The matrix A look like:

$$\begin{pmatrix}
0 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4/6 & 0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2/6 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/6 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/6 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/6 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/6 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 1/6 & 1/6 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 1/6 & 1/6 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 1/6 & 1/6 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 1/6 & 0 & 1/6 & 1/6 & 1/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 2/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 0 & 0 & 1/6 & 1/6 & 3/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 4/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 5/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

We can justify that this matrix is correct. For example, if we start from state 25, then for roll 1, 2, 3, 4, 5, 6, our added result would be 26, 27, 28, 29, 30, 31. All five results are either equals or surpasses our limit 26. We can see from the above matrix that:

$$A_{26,27} = 1 \ (all\ indexes\ added\ 1\ since\ we\ start\ from\ State\ 0)$$

The rest entries can be checked with the same reasoning.

For limit being 26, we can get:

$$Expected\ Number = \frac{3159}{209} \approx 15.1148382307449$$

This means the game takes about 15.1 rolls until our score reaches or exceeds 26 on average. Some questions can be asked in here. For example, what is the probability that at least half the expected number of rolls are needed to get to 26? Half of the expected number in this case is 8. Notice that our game is totally random, if we are unlucky, we may never reach the limit 26. There is no upper bound for the rolls we need to end the game.

We can think this problem in a different way. Instead of trying to find all the probabilities that the game ends at least half of the expected rolls, we can find the probability the game ends on or before we reach half of the expected rolls. That is, we need to fine $A^7_{1,27}$, which we learned about it is the probability the game has ended on or before the 7th roll of the die. Then we can have 1 minus $A^7_{1,27}$ which gives what we want.

In MATLAB, we can write the following code:

```
% Probability of at least half of expected number of rolls needed to end.

temp = A^7;
prob = 1 - temp(1, limit + 1);
```

$$Probaility \approx 0.893407778920896$$

This means the probability that at least half the expected number of rolls are needed to get to 26 is about 0.8934.

III. Investigation:

There is a relation between the limit and the expected number. We want to investigate such relation. I have generated the transition matrix A for different limit, got the expected number to each of the limit with the code below:

```
% Generate the transition matrix

limit = [10, 20, 26, 40, 50, 70, 100, 150, 200, 250, 300, 350, 400, 500, 750, 1000];
expect_num = zeros(length(limit), 1);
idx = 1;

for i = limit
    A = transMatrix(i);
    % Generate Q, N, and Expected Number
    Q = A(1:i, 1:i);
    N = inv(eye(i) - Q);
    expect_num(idx) = sum(N(1, :));
    idx = idx + 1;
end
```

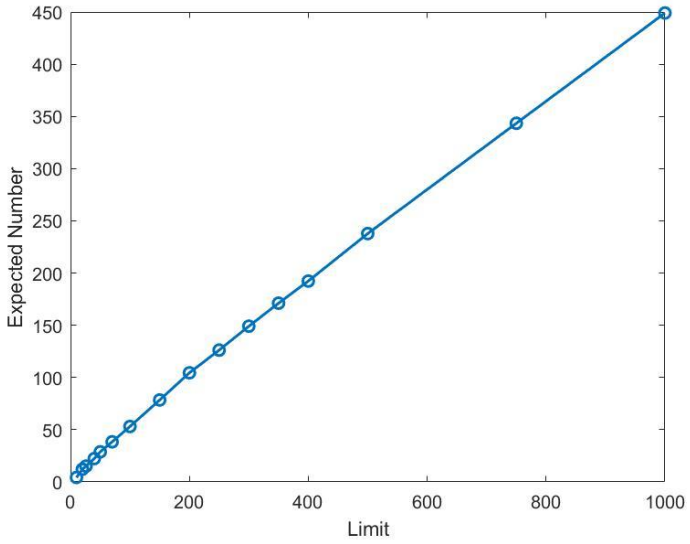
The table looks like:

Limit	Expected Number
10	4.24068922886725
20	12.1363729277693
26	15.1148382307449
40	22.3604528903354
50	28.79490620831
70	38.3903159680711
100	53.0984671621918
150	78.449850659704
200	104.454875653201
250	126.202335249332
300	149.113288401965
350	171.181594324969
400	192.337011993821
500	237.781746067832
750	343.415591411981
1000	449.015889832685

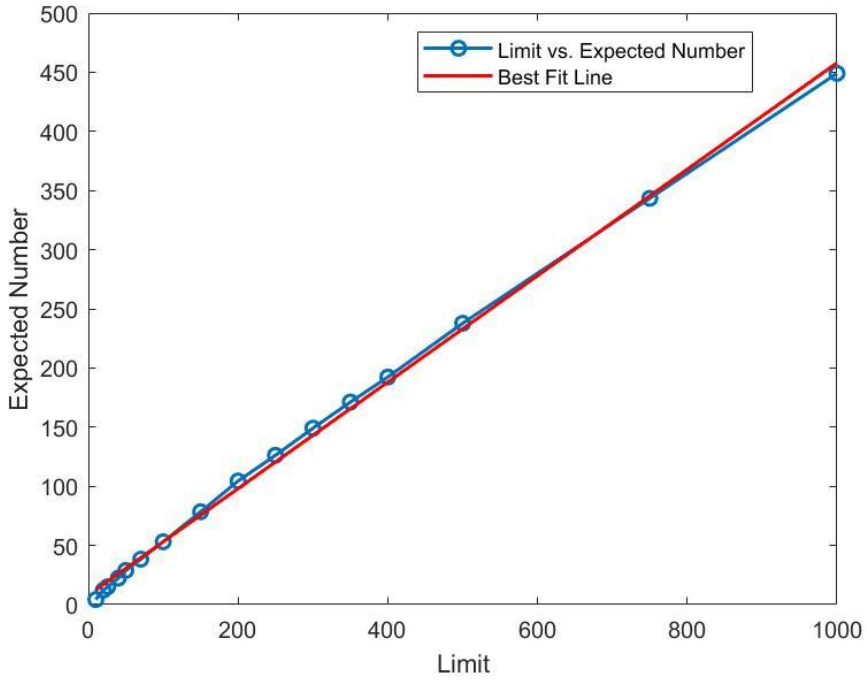
To see what kind of relation is between the limit and the expected number, we can plot it:

```
x = limit;
y = expect_num;
plot(x, y, 'LineWidth', 1.5); xlabel('Limit'); ylabel('Expected Number');
```

The plot looks like:



It seems like there is a linear relation between the limit and the expected number. To make sure, we add the best fit linear line:



We can see that the beft fit line works very well in the plot. To get the exact equation between the limit and the expected number, we run the following MATLAB code:

```
x = limit;
y = expect_num;
coeff_lin = polyfit(x, y, 1);
```

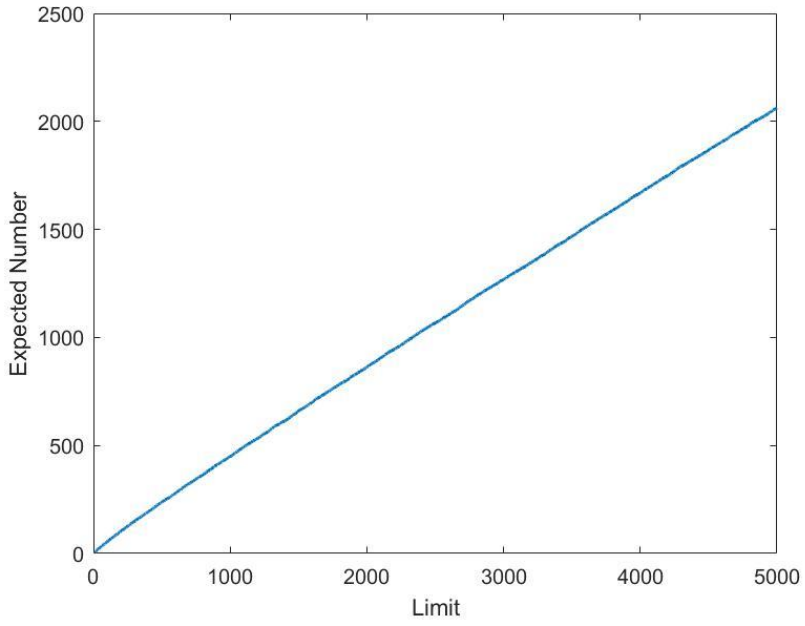
This gives us the slope of the best fit line is 0.4498, with y-intercpt being 8.1143, we have:

$$Expected\ Number = 0.4498 \cdot Limit + 8.1143$$

One thing to notice is that when limit is 0, the above linear equation would give us the expected number being 8.1143 which is not correct. This might be our sample size is limited and fairly separated. If we take a larger set of limit values with a more consistet inverval between the different limit values, we may be able to get a more accurate slope and y-intercept. However, there are too many uncertainties in here, so we want to investigate more.

With the following MATLAB code, we try to generate the expected number for a more wide and even separated limit set, then plot it:

```
% For a larger set of limits with increment of 2, range from 10 to 5000

limit = 10:2:5000;
expect_num = zeros(length(limit), 1);
idx = 1;

for i = limit
    A = transMatrix(i);
    % Generate Q, N, and Expected Number
    Q = A(1:i, 1:i);
    N = inv(eye(i) - Q);
    expect_num(idx) = sum(N(1, :));
    idx = idx + 1;
end

% Plot it

x = limit;
y = expect_num;
plot(x, y,'LineWidth', 1.5); xlabel('Limit'); ylabel('Expected Number');
```

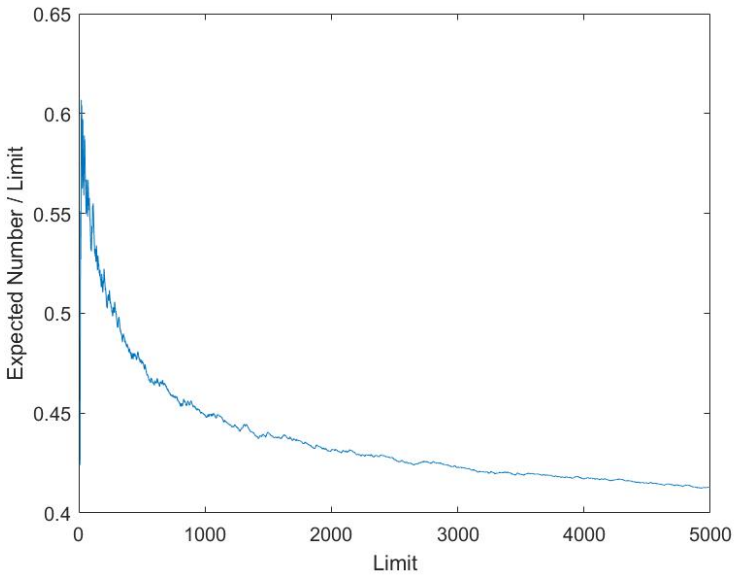


We can see that even with a set of limits with consistent 2 increment from range 10 to 5000, our plot looks like the plot made previously. We will investigate what does the plots look like for:

$$\frac{Expected\ Value}{Limit} \text{ vs. } Limit, \frac{Expected\ Value^2}{Limit} \text{ vs. } Limit, \text{ and } \frac{\log(Expected\ Value)}{\log(Limit)} \text{ vs. } Limit$$

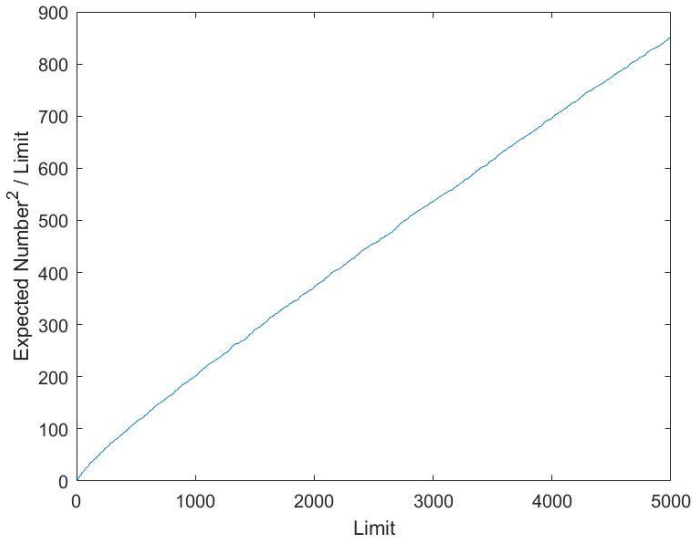
We can generate the plot with the code:

```
% Plot

tt = zeros(length(limit), 1);

for j = 1:length(limit)
    tt(j) = sqrt(expect_num(j)) / limit(j);
end

plot(limit, tt); xlabel('Limit'); ylabel('Expected Number / Limit');

tt2 = zeros(length(limit), 1);

for j = 1:length(limit)
    tt2(j) = (expect_num(j)^2) / limit(j);
end

plot(limit, tt2); xlabel('Limit'); ylabel('Expected Number^2 / Limit');

tt3 = zeros(length(limit), 1);

for j = 1:length(limit)
    tt3(j) = log(expect_num(j)) / log(limit(j));
end

plot(limit, tt3); xlabel('Limit'); ylabel('log(Expected Number) / log(Limit)');
```

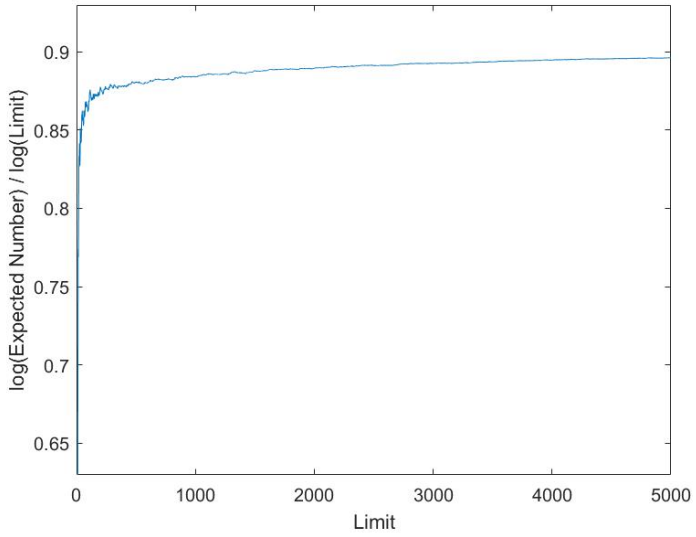


From $\frac{Expected\ Value}{Limit}$ vs. $Limit$ plot, we can see that as $Limit$ increases, $\frac{Expected\ Value}{Limit}$ decreases.

If the relation between expected value and limit is linear, then the plot would be a straight horizontal line. So, it is not linear.



Same here for $\frac{Expected\ Value^2}{Limit}$ vs. $Limit$ plot, when limit increase, $\frac{Expected\ Value^2}{Limit}$ also increases instead of being constant. So, this is not the right one.



However, if we look at the $\frac{\log(Expected\ Value)}{\log(Limit)}$ vs. $Limit$ plot, for large limit, as limit increases, $\frac{\log(Expected\ Value)}{\log(Limit)}$ tends to be a constant. In MATLAB, if we take a look at $tt3$ which we generated use the code in the previous page, we take the mean value which gives our a constant about 0.88.

If we use the constant to write our equation, we can a power function:

$$Expected\ Number = Limit^{0.88}$$

To compare the equation which we generated for the linear relation:

$$Expected\ Number = 0.4498 \cdot Limit + 8.1143$$

For limit being 26, the power function evaluates 17.5864, the linear function evaluates 19.80. We can see that 18.53 is closer to 15.11 (our calculated expected number). Another example, for limit equals 500, the power function evaluetes 237.1885, where the linear function evaluets 233.0143. Our calculated value is 237.7817. Again, the power function gives a better evaluation depend on our computed values.

Considering with the investigations we have made so far, it would better to say that the expected number and the limit have a power relation with a power constant 0.88 instead of a linear relation we claimed earlier. However, the power constant 0.88 can still be examined if we have a much larger sample size than 5000, but it does give us a better understanding of the relation.

IV. Simulation:

To be more confident with the results we get in the previous sections, we will implement a simulation of the game. The code below simulates the random process of our game for limit is 26 for 100000 tries.

```
limit = 26;
games = 1:100000;
throws = zeros(1, 100000);

for i = games
    score = 0;
    while score < limit
        throws(i) = throws(i) + 1;
        % random rolls
        roll = randi(6);
        if isprime(score + roll)
            if score + roll >= limit
                break;
            elseif score == 0
                score = roll;
            else
                score = score - roll;
                if score < 0
                    score = 0;
                end
            end
        else
            score = score + roll;
        end
    end
end
```

Then, for the example where our limit is set to 26, we can get the average throws, and the probability that at least half the expected number of rolls are needed to end the game:

```
% Average throws

average_throws = mean(throws);

% Verify the probabily we get in Section II.

verify = 1 - sum(throws <= 7) / length(throws);
```

We can see that the results are very close with the ones we have:

$$Expected\ Number\ (Stimulation) \approx 15.09011$$

$$Probility\ (Stimulation) \approx 0.89387$$

To see if this holds for other limits we set, here is a table to summarize:

Limit	Expected Number (Denote EN)	Expected Number (Simulation) (Denote ENS)	Percentage Difference (%) $\frac{\lvert EN - ENS \rvert}{[\frac{(EN + ENS)}{2}]} \times 100$
10	4.24068922886725	4.24028	0.00965052121308387
20	12.1363729277693	12.08848	0.395403248986632
26	15.1148382307449	15.1479	0.218498200678724
40	22.3604528903354	22.3883	0.124459824535653
50	28.79490620831	28.78236	0.0435804237895384
70	38.3903159680711	38.4276	0.0970711883004204
100	53.0984671621918	53.08283	0.0294537034481986
150	78.449850659704	78.38338	0.0847660402382382
200	104.454875653201	104.4293	0.0244878800615133
250	126.202335249332	126.32057	0.0936427929590177
300	149.113288401965	149.14451	0.020935981021969
350	171.181594324969	171.17715	0.00259629703816171
400	192.337011993821	192.37266	0.0185324200423337
500	237.781746067832	237.78577	0.00169226535949443
750	343.415591411981	343.49894	0.0242675279697068
1000	449.015889832685	449.19338	0.039520894133675

We calculated the average percentage difference which is about 0.0767849506110225%. So, on average, there is about 0.077% percentage difference between the values we had, and the value acquired from stimulation, which is tolerable in this case since the numbers represents the rolls which should all be integers. We can see from the table above that all the integer part are identical between the two value sets. Even though the difference is small, it still does not prove or verify that our result is correct. But the simulation gives us confidence that the errors in our computation is small, so we feel more confident that the calculated value is at least close to the truth.

V. Conclusion and Comment:

We can see that Markov Chain is a very useful tool when we want to study random events when we know the probabilities of each transition state (from one state to another). Through manipulation with the transition matrix, we can get a reasonable and fairly accurate prediction for different cases once we have generated and understanded the pattern of a problem. But it may not suit for super large state spaces, for example if our limit for the dice game is set to 100000, then the transition matrix would be 100001 by 100001 which may be very computational time consuming. For a smaller set of state space problems, we can use Markov Chain and the transition matrix to get some very useful information.