Absorbing States and z-Transforms

Absorbing States

Markov chains solve problems of the form

$$x(k+1) = A\,x(k)$$

along with an initial condition

$$x(k=0) = X_0$$

In the case of three people tossing a ball in our last lecture, the ball keeps moving around and never ends up anywhere in particular. In some cases, there is a definite end point.

For example, take the case of playing a game where the game is over once one team us up by two games. The state-transition matrix for this game was

$$X(k+1) = \begin{bmatrix} 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0 \\ 0 & 0.3 & 0 & 0.7 & 0 \\ 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 1 \end{bmatrix} X(k)$$

where these states are defined as

$$X = \begin{bmatrix} \text{up 2 games (player A wins)} \\ \text{up 1 game} \\ \text{tied} \\ \text{down 1 game} \\ \text{down 2 games (player B wins)} \end{bmatrix}$$

Here, if you encounter state 1 or 5, the game ends. This is denoted in the state-transition matrix with a 1.00 in a row (once player A has won, he/she has always won). This is called an absorbing state.

If you have an absorbing state, as time goes to infinity you will always wind up there. This system has two absorbing states (player A wins and player B wins). The value of X determines the probability of getting to each of these states.

To determine the steady-state solution, you could

- Play the game a large number of times

Option 1: Find the steady-state solution

$$x(k+1) = Ax(k) = x(k)$$

$$(A - I)x(k) = 0$$

This doesn't help in this case due to the absorbing state. If you try to solve, you get

```
A - eye(5,5)

         0         0         0         0         0
    0.7000   -1.0000    0.3000         0         0
         0    0.7000   -1.0000    0.3000         0
         0         0    0.7000   -1.0000    0.3000
         0         0         0         0         0
```

$$\begin{bmatrix} 0 & 0.7 & 0 & 0 & 0 \\ 0 & -1 & 0.7 & 0 & 0 \\ 0 & 0.3 & -1 & 0.7 & 0 \\ 0 & 0 & 0.3 & -1 & 0 \\ 0 & 0 & 0 & 0.3 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = 0$$

This has the solution

$$X(\infty) = \begin{bmatrix} a \\ 0 \\ 0 \\ 0 \\ e \end{bmatrix}$$

which simply tells you that eventually someone wins.

Option 2: Eigenvectors.

The eigenvalues and eigenvectors tell you

- How the system behaves (eigenvalues) and
- What behaves that way.

For steady-state solutions, we care about the eigenvectors associated with the eigenvalues at 1.0000

```
[M,V] = eig(A)

    1.0000         0   -0.8033    0.2846   -0.5384
         0         0    0.4039   -0.6701    0.7692
         0         0    0.3739    0.6203   -0.0000
         0         0    0.1731   -0.2872   -0.3297
         0    1.0000   -0.1476    0.0523    0.0989

V:  1.0000    1.0000    0.6481   -0.6481         0
```

There are two absorbing states: hence there are two eigenvalues at 1.000

The eigenvectors tell you that eventually,

- A wins (first eigenvector), or
- B wins (second eivenvector).

Again, this doesn't rally help.

Option 3: Play the game a large number of times (100 times).

In Matlab:

```
A = [1,0,0,0,0;0.7,0,0.3,0,0;0,0.7,0,0.3,0;0,0,0.7,0,0.3;0,0,0,0,1]'

    1.0000    0.7000         0         0         0
         0         0    0.7000         0         0
         0    0.3000         0    0.7000         0
         0         0    0.3000         0         0
         0         0         0    0.3000    1.0000

A^100

    1.0000    0.9534    0.8448    0.5914         0
         0         0    0.0000         0         0
         0    0.0000         0    0.0000         0
         0         0    0.0000         0         0
         0    0.0466    0.1552    0.4086    1.0000
```

Note that A^n will be multiplied by the initial condition. This means the row of A tells you the probability of

- A winning (first row) and
- B winning (last row)

The columns tell you the probability if you offer odds:

- Column 1: Player A starts with a +2 game advantage (player A always wins)
- Column2: Player A starts with a +1 game advantage (A wins 95.34% of the time)
- Column 3: Player A starts with a +0 game advantage (A wins 84.48% of the time)
- Column 4: Player B starts with a +1 game advantage (A wins 59.14% of the time)
- Column 5: Player B starts with a +2 game advantage (B always wins)

The nice thing about this approach is you can see the affect of giving odds at the start of the match on the eventual winner.

Good Money After Bad

In gambling, there is a saying: *Good Money After Bad.* This means that if you start losing, some people will keep gambling to try to recoup their losses. This tends to result in the person going broke: they are risking money they have (good money) to recoup money they've lost (bad money).

For example, suppose you play a game of chance.

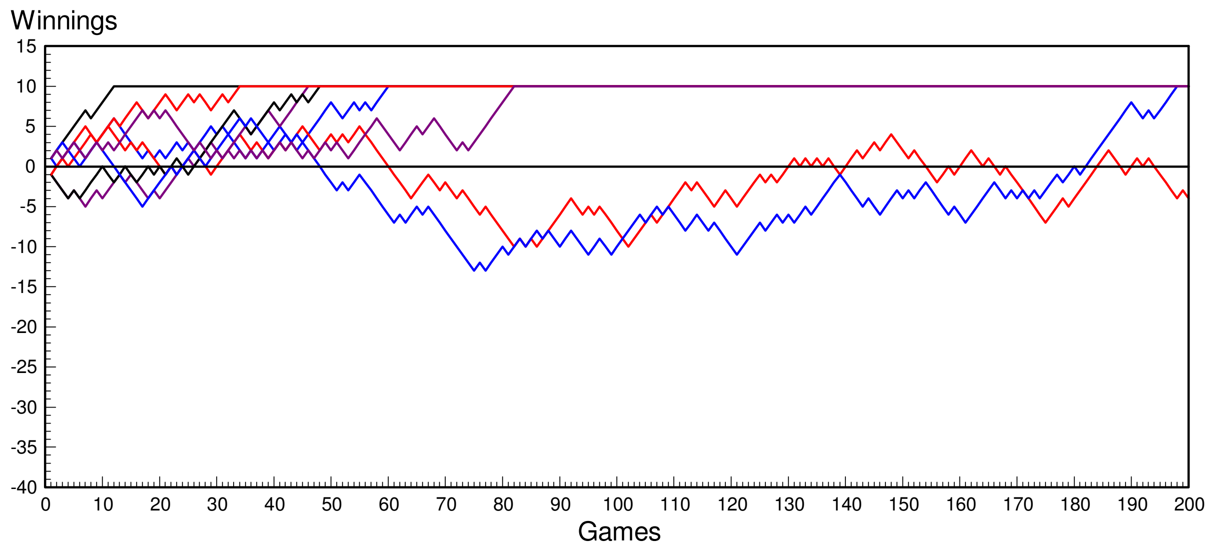
- 49% of the time you win and earn $1.
- 51% of the time you lose and lose $1.

Suppose you plan on stopping once you are up $10. This is an absorbing state - meaning you will (in theory) always wind up there. It sounds like you will always wind up plus $10.

In Matlab, you can test this theory with a Monte Carlo simulation.

- Keep playing until you are up $10.

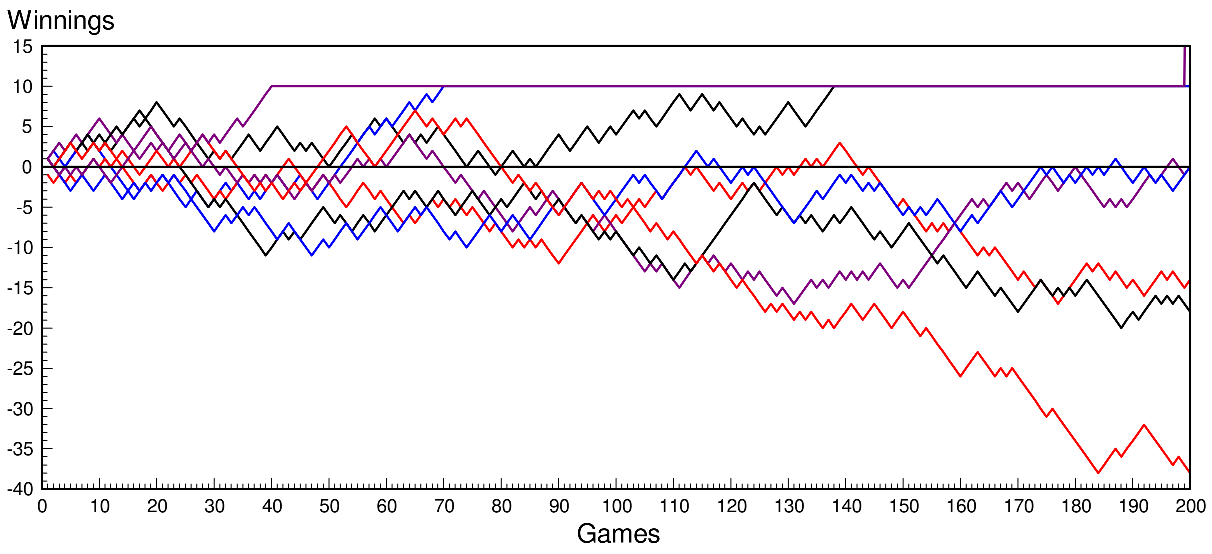
```
% game of chance

X = 0;   % winnings
n = 0;   % number of games

while (X < 10)
    n = n + 1;
    if (rand < 0.49)
        X = X + 1;
    else
        X = X - 1;
    end
    disp([n,X]);
end

X
```



Winnings after n games: Stop when you reach $10

Most of the time you end up at +10 dollars and the game stops. In theory, eventually you will always end up at +$10.

Now change the problem slightly: instead of having a 53% change of winning any game, it's onl 47%



Winnings after n games with a 47% chance of winning any given game.

Gain, fairly often you end up +$10 and quit the game. Sometimes you end up just getting further and further in the hole.

The creates a conflict:

- If you only have one absorbing state, you should always wind up at that absorbing state.
- With Monte Carlo simulations, this usually happens, but not always.

A better model would be to add a second absorbing state:

- Once you are down $100, the house no longer accepts your money.

In matlab, you can simulate this as

```
% game of chance

Win = 0;

for i=1:10000

X = 0;   % winnings
n = 0;   % number of games

while ((X < 10) & (X > -100))
     n = n + 1;
     if (rand < 0.49)
         X = X + 1;
```

```
        else
            X = X - 1;
        end
       % disp([n,X]);
    end

    if(X == 10)
        Win = Win + 1;
    end

    end
    Win
```

In 10,000 games

- 6,647 times you're up $10
- 3,353 times you're down $100

If you're losing, it's best to cut your losses and walk away.

The net result is that you are up $10 1000 times out of 1000 trials.

z-Transforms

Suppose you want to determine the probability of player A winning after k games. This is where z-transforms shine.

z-Transforms designed to determine the time response of a discrete-time system

$$X(k+1) = AX(k) + BU(k)$$

$$Y = CX(k)$$

Here,

- X(k) are the states
- U(k) is the input
- Y(k) is what you're measuring.

If U(k) is an impulse function, you get the impulse response:

$$zX = AX + B$$

$$Y = CX$$

This is what we want with Markov chains, only

- B is the initial condition: X(0)
- C tells you which state you want to look at.

For example, for the problem of winning by 2 games,

```
A =

    1.0000    0.7000         0         0         0
         0         0    0.7000         0         0
         0    0.3000         0    0.7000         0
         0         0    0.3000         0         0
         0         0         0    0.3000    1.0000

X0 = [0;0;1;0;0]

     0
     0
     1
     0
     0
```

To determine the probability of A winning, look a the first state (A is up 2 games):

```
C = [1,0,0,0,0]

C =     1     0     0     0     0
```

You can now find the Y(z) (or the impulse response)

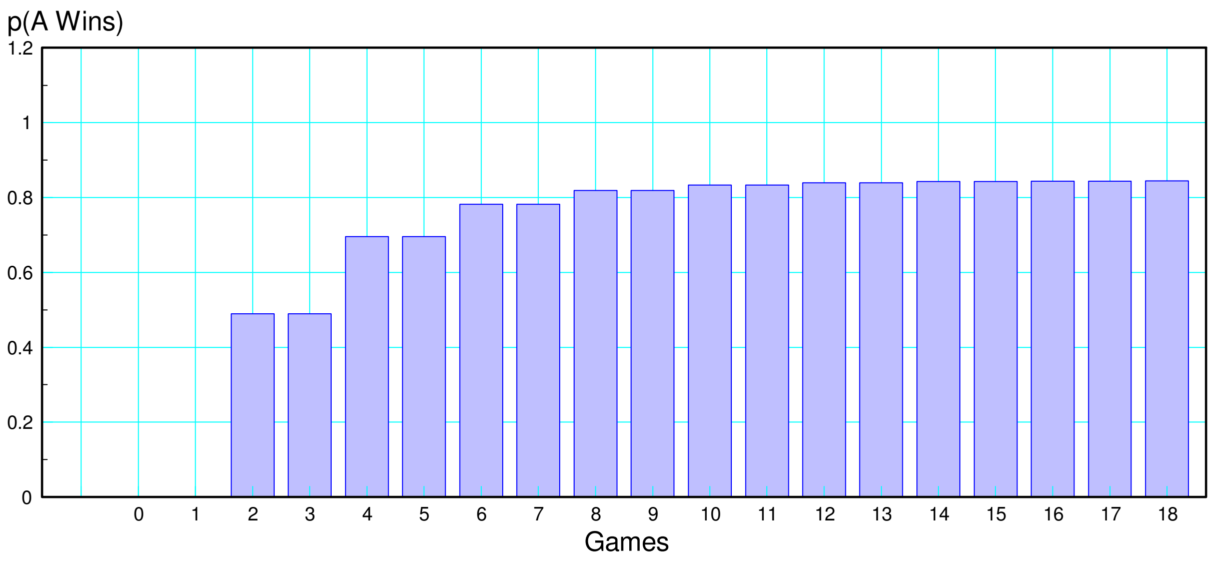
```
G  = ss(A,X0,C,0,1);

tf(G)

                      0.49 z
------------------------------------------
z^4 - z^3 - 0.42 z^2 + 0.42 z + 1.821e-018

sampling time (seconds): 1

zpk(G)

             0.49 z
-----------------------------
z (z-1) (z-0.6481) (z+0.6481)

Sampling time (seconds): 1
```

From our last lecture, you need to multiply by z to get Y(z). This tells you that

$$Y(z) = \left(\frac{0.49z}{(z-1)(z-0.6481)(z+0.6481)}\right)$$

The time response is from the *impulse* function

```
y = impulse(G)

y =      0
         0
         0
    0.4900
    0.4900
    0.6958
```



You can also find the explicit function for y(k) using z-transfoms.

From Matlab, the z-transform for Y(z) is:

$$Y(z) = \left(\frac{0.49}{(z-1)(z-0.6481)(z+0.6481)}\right)$$

Since the denominator is 3rd order and the numerator is 0th order, the system has a delay of 3 (A can't win until after the 3rd game). Multiply both sides by z^3

$$z^3 Y = \left(\frac{0.49z^3}{(z-1)(z-0.6481)(z+0.6481)}\right)$$

Factor out a z (the z-transform table has a z in the numerator)

$$z^3 Y = \left(\frac{0.49z^2}{(z-1)(z-0.6481)(z+0.6481)}\right) z$$

Do partial fractions

$$z^3 Y = \left(\left(\frac{0.8449}{z-1}\right) + \left(\frac{0.4512}{z-0.6481}\right) + \left(\frac{0.0963}{z+0.6481}\right)\right) z$$

Take the inverse z-transform

$$z^3 y(k) = \left(0.8449 + 0.4512\ (0.6481)^k + 0.0963\ (-0.6481)^k\right)\ u(k)$$

Divide by z^3 (delay 3 samples)

$$y(k) = \left(0.8449 + 0.4512\ (0.6481)^{k-3} + 0.0963\ (-0.6481)^{k-3}\right)\ u(k-3)$$