Simulation

Daniel J. Eck

Background

This lecture is meant to supplement Chapter 9 in your textbook.

We already performed an analysis on the value of baseball events from seasonal data.

Now we look at simulation as a means for understanding different characteristics of a half-inning.

Simulating a half inning

We will focus on simulating the events in a half inning using a special probability model called a Markov chain.

By simulating many half-innings using this model, one gets a basic understanding of the pattern of run scoring.

Markov Chain basics

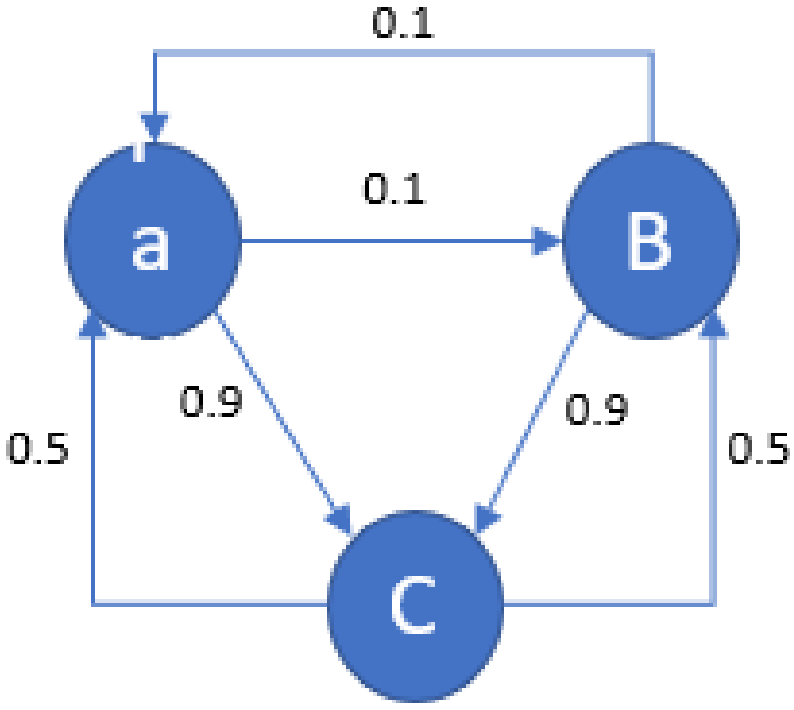
A Markov chain is a special model describing movement between states.

In a (discrete) Markov chain, a matrix of transition probabilities is used to describe how one moves between the different states.

In a Markov chain, one assumes that the probability of moving to a new state only depends on the current state.

One learns how to move to a new state from the current state by examining the rows of the transition probability matrix.

In a Markov chain, there are two states: transition states and absorbing states. Once one lands in an absorbing state, one cannot return to a transition state.



$$P = \begin{bmatrix} 0 & 0.1 & 0.9 \\ 0.1 & 0.0 & 0.9 \\ 0.5 & 0.5 & 0.0 \end{bmatrix}$$

(image credit)

The definitions and properties of discrete-time Markov chain process

- **Definition 2.1** *The random sequence* $(X_n, n \in \mathbb{N})$ *is a Markov chain if for all* $x_0, x_1, \ldots, x_n \in I$:

$$P\left(X_n = j_n \middle| X_0 = j_0, X_1 = j_1, \ldots, X_{n-1} = j_{n-1}\right) = P\left(X_n = j_n \middle| X_{n-1} = j_{n-1}\right)$$

(*provided this probability has meaning*).

- **Definition 2.2** *A Markov chain* $(J_n, n \geq 0)$ *is homogeneous if the probabilities do not depend on n and is non-homogeneous in the other cases.*

For the moment, we will only consider the homogeneous case for which we write:

$$P\left(X_n = j \middle| X_{n-1} = i\right) = p_{ij}$$

and we introduce the matrix **P** defined as: $P = \lfloor p_{ij} \rfloor$

(image credit)

Connecting Markov chains to baseball context

A state is a base out configuration. There are 24 possible base out configurations.

We will add the 3 out state as an absorbing state, so there are 25 total states.

Example of transition state:

- Suppose that there are runners on 1st and 2nd with 1 out.
- Based on the outcome of the plate appearance, the state can change.
- For example, the batter can hit a single, the runner on 2nd scores and the runner on 1st moves to 3rd.
- The new state is now runners on 1st and 3rd with 1 out.

Any baseball events that happened before the current state do not affect the transition probabilities.

Markov chain for the 2016 season

We need to know the frequencies of transitions from the different base outs states to other such states in order to construct the transition probability matrix.

We will do this for the 2016 season. Attention is restricted to batting plays and complete innings. Non-batting plays such as stolen bases, caught stealing, wild pitches, or passed balls are ignored.

```
library(tidyverse)
```

```
fields = read_csv("../../fields.csv")
dat2016 = read_csv("../../all2016.csv",
                   col_names = pull(fields, Header),
                   na = character())
```

We create `HALF.INNING` as a unique identification for each half-inning in each baseball game.

The new variable `RUNS.SCORED` gives the number of runs scored on each play.

```
dat2016 = dat2016 %>%
  mutate(RUNS  = AWAY_SCORE_CT + HOME_SCORE_CT,
         HALF.INNING = paste(GAME_ID, INN_CT, BAT_HOME_ID),
         RUNS.SCORED = (BAT_DEST_ID > 3) + (RUN1_DEST_ID > 3) +
           (RUN2_DEST_ID > 3) + (RUN3_DEST_ID > 3))

half_innings = dat2016 %>%
  group_by(HALF.INNING) %>%
  summarize(Outs.Inning = sum(EVENT_OUTS_CT),
            Runs.Inning = sum(RUNS.SCORED),
            Runs.Start = first(RUNS),
            MAX.RUNS = Runs.Inning + Runs.Start)
```

We create the variable `STATE` which gives the runner locations and outs at the beginning of each play.

The variable `NEW.STATE` encodes the base out state at the end of the play.

```
dat2016 = dat2016 %>%
  inner_join(half_innings, by = "HALF.INNING") %>%
  mutate(BASES = paste(ifelse(BASE1_RUN_ID > '', 1, 0),
                       ifelse(BASE2_RUN_ID > '', 1, 0),
                       ifelse(BASE3_RUN_ID > '', 1, 0), sep = ""),
         STATE = paste(BASES, OUTS_CT),
         NRUNNER1 = as.numeric(RUN1_DEST_ID == 1 | BAT_DEST_ID == 1),
         NRUNNER2 = as.numeric(RUN1_DEST_ID == 2 | RUN2_DEST_ID == 2 | BAT_DEST_ID == 2),
         NRUNNER3 = as.numeric(RUN1_DEST_ID == 3 | RUN2_DEST_ID == 3 |
                               RUN3_DEST_ID == 3 | BAT_DEST_ID == 3),
         NOUTS = OUTS_CT + EVENT_OUTS_CT,
         NEW.BASES = paste(NRUNNER1, NRUNNER2, NRUNNER3, sep = ""),
         NEW.STATE = paste(NEW.BASES, NOUTS))

## restrict attention to state changes and complete innings; remove non-batting plays.
## remove base out context when absorbing state is reached,
## and change the name of this state.
dat2016 = dat2016 %>% filter((STATE != NEW.STATE) | (RUNS.SCORED > 0))
dat2016C = dat2016 %>% filter(Outs.Inning == 3, BAT_EVENT_FL == TRUE)
dat2016C = dat2016C %>% mutate(NEW.STATE = gsub("[0-1]{3} 3", "3", NEW.STATE))
```

Computing the transition probabilities

With the `STATE` and `NEW.STATE` variables defined, we can compute the frequencies of all possible transitions.

```
T_matrix = dat2016C %>%
  select(STATE, NEW.STATE) %>%
  table()

P_matrix = prop.table(T_matrix, 1)

## add a row to give meaning to the absorbing state
P_matrix = rbind(P_matrix, c(rep(0, 24), 1))
```

Example of the transition matrix using the “000 0” state.

The `as.tibble` and `gather` functions are used to display the probabilities vertically.

```
P_matrix %>%
  as.tibble(rownames ="STATE") %>%
  filter(STATE == "000 0") %>%
  gather(key = "NEW.STATE", value = "Prob", -STATE) %>%
  filter(Prob > 0)
```

```
## # A tibble: 5 x 3
##   STATE NEW.STATE    Prob
##   <chr> <chr>       <dbl>
## 1 000 0 000 0     0.0334
## 2 000 0 000 1     0.676
## 3 000 0 001 0     0.00563
## 4 000 0 010 0     0.0503
## 5 000 0 100 0     0.235
```

Simulating the Markov chain

We will use Markov chains to obtain the distribution of runs scored in a half-inning.

The first step is to construct a matrix giving the runs scored in all possible transitions between states.

Let N_{runners} denote the number of runners in a state, and O denote the number of outs. For a batting play $(s \to t)$, the number of runs scored is equal to,

$$R = (N_{\text{runners}}^{(s)} + O^{(s)} + 1) - (N_{\text{runners}}^{(t)} + O^{(t)}).$$

We define a function which takes a state as input and returns the number of runs and outs.

```
count_runners_out = function(s){
  s %>% str_split("") %>% pluck(1) %>% as.numeric() %>% sum(na.rm = TRUE)
}

runners_out = sapply(row.names(T_matrix),
                     count_runners_out)[-25]
```

The `outer` functions with the "-" operation performs the runs *R* calculation for all possible pairs of states.

```
R = outer(runners_out + 1, runners_out, FUN = "-")
names(R) = names(T_matrix)[-25]
R = cbind(R, rep(0,24))
```

We are now ready to simulate a half-inning of baseball using a new function.

The inputs of this function are:

- probability transition matrix `P`
- the runs matrix `R`
- the starting state `s`

The output is the the number of runs scored in the half-inning

```
set.seed(430)
simulate_half_inning = function(P, R, start = 1){
  s = start
  path = NULL
  runs = 0
  while(s < 25){
    s.new = sample(1:25, size = 1, prob = P[s, ])
    path = c(path, s.new)
    runs = runs + R[s, s.new]
    s = s.new
  }
  runs
}
```

Let's try it out

```
B = 1e5
system.time({
  RUNS = replicate(B, simulate_half_inning(P_matrix, R))
})
```

```
##    user  system elapsed
##   1.931   0.039   1.971
```

Some inferences:

```
## table of runs
table(RUNS)
```

```
## RUNS
##     0     1     2     3     4     5     6     7     8     9    10    11    13
## 73625 14266  6785  3099  1303   555   225    86    32    17     5     1     1
```

```
## chance of scoring 5 or more runs
sum(RUNS >= 5) / B
```

```
## [1] 0.00922
```

```
## average number of runs
mean(RUNS)
```

```
## [1] 0.47555
```

Compare with Runs Expectancy

```
RUNS.j = function(j){
  mean(replicate(B, simulate_half_inning(P_matrix, R, j)))
}
```

```
library(doMC)
library(parallel)
registerDoMC(cores=detectCores()-2)
## important for reproducibility
RNGkind(kind = "L'Ecuyer-CMRG")
system.time({
  RE_bat = foreach(j = 1:24) %dopar% RUNS.j(j) %>%
    unlist() %>%
    matrix(nrow = 8, ncol = 3, byrow = TRUE,
           dimnames =list(c("000","001","010","011",
                            "100","101","110","111"),
                          c("0 outs", "1 out", "2 outs")))
})
```

```
##    user  system elapsed
##  34.641   1.171   4.596
```

```
round(RE_bat, 2)
```

```
##     0 outs 1 out 2 outs
## 000   0.48  0.25   0.10
## 001   1.31  0.93   0.34
## 010   1.10  0.64   0.30
## 011   1.91  1.31   0.49
## 100   0.84  0.49   0.20
## 101   1.71  1.13   0.44
## 110   1.41  0.88   0.41
## 111   2.21  1.48   0.67
```

We see that the Markov chain simulation does well, but it misses on the run scoring value of non-batting plays.

```
RE = matrix(c(0.47, 0.25, 0.10, 1.45, 0.94, 0.32, 1.06, 0.65,
              0.31, 1.93, 1.34, 0.54, 0.84, 0.50, 0.22, 1.75,
              1.15, 0.49, 1.41, 0.87, 0.42, 2.17, 1.47, 0.76),
              8, 3, byrow=TRUE)
round(RE - RE_bat, 2)
```

```
##     0 outs 1 out 2 outs
## 000  -0.01  0.00   0.00
## 001   0.14  0.01  -0.02
## 010  -0.04  0.01   0.01
## 011   0.02  0.03   0.05
## 100   0.00  0.01   0.02
## 101   0.04  0.02   0.05
## 110   0.00 -0.01   0.01
## 111  -0.04 -0.01   0.09
```

Beyond run expectancy

We can learn about the movement through the base out states. For example, let's investigate the typical states after 3 PAs to start a half-inning

```
P_matrix_3 = P_matrix %*% P_matrix %*% P_matrix

P_matrix_3 %>% as_tibble(rownames = "STATE") %>%
  filter(STATE == "000 0") %>%
  gather(key = "NEW.STATE", value = "Prob", -STATE) %>%
  arrange(desc(Prob)) %>%
  head()
```

```
## # A tibble: 6 x 3
##   STATE NEW.STATE   Prob
##   <chr> <chr>      <dbl>
## 1 000 0 3          0.372
## 2 000 0 100 2      0.241
## 3 000 0 110 1      0.0815
## 4 000 0 010 2      0.0739
## 5 000 0 000 2      0.0529
## 6 000 0 001 2      0.0286
```

Transition probabilities for individual teams

We construct a data frame `Team.T` giving the transitions for each team.

```
dat2016C = dat2016C %>%
  mutate(HOME_TEAM_ID = str_sub(GAME_ID, 1, 3),
         BATTING.TEAM = ifelse(BAT_HOME_ID == 0,
                               AWAY_TEAM_ID, HOME_TEAM_ID))

Team.T = dat2016C %>%
  group_by(BATTING.TEAM, STATE, NEW.STATE) %>%
  count()

Team.T %>% filter(BATTING.TEAM == "SLN")
```

```
## # A tibble: 191 x 4
## # Groups:   BATTING.TEAM, STATE, NEW.STATE [191]
##    BATTING.TEAM STATE NEW.STATE     n
##    <chr>        <chr> <chr>     <int>
##  1 SLN          000 0 000 0        64
##  2 SLN          000 0 000 1      1016
##  3 SLN          000 0 001 0        12
##  4 SLN          000 0 010 0        96
##  5 SLN          000 0 100 0       338
##  6 SLN          000 1 000 1        37
##  7 SLN          000 1 000 2       732
##  8 SLN          000 1 001 1         8
##  9 SLN          000 1 010 1        56
## 10 SLN          000 1 100 1       263
## # i 181 more rows
```

Adjustments for rare transitions need to be made to get reliable predictions.

Let p^{TEAM} denote a team's transition probability, and let p^{ALL} be the marginal transition probabilities across all teams. Then, a reliable estimate is given by

$$p^{\text{EST}} = \frac{n}{n+K} p^{\text{TEAM}} + \frac{K}{n+K} p^{\text{ALL}}$$

where n is the number of transitions and K is a smoothing count. Here, we will let $K = 1274$ (reasons beyond scope of course).

Let's demonstrate the smoothing with respect to the Cardinals 2016 seasonal results in the "100 2" base out state.

```
Team.T.S = dat2016C %>% filter(STATE == "100 2") %>%
  group_by(BATTING.TEAM, STATE, NEW.STATE) %>% tally()

SLN.Trans = Team.T.S %>% filter(BATTING.TEAM == "SLN") %>%
  mutate(p = n / sum(n))

All.Trans = dat2016C %>% filter(STATE == "100 2") %>%
  group_by(NEW.STATE) %>% tally() %>%
  mutate(p = n / sum(n))

SLN.Trans %>% inner_join(All.Trans, by = "NEW.STATE") %>%
  mutate(p.EST = n.x / (1274 + n.x) * p.x + 1274 / (1274 + n.x) * p.y) %>%
  ## needs to be normalized
  mutate(p.EST = p.EST / sum(p.EST)) %>%
  select(BATTING.TEAM, NEW.STATE, p.x, p.y, p.EST)
```

```
## # A tibble: 8 x 6
## # Groups:   BATTING.TEAM, STATE [1]
##   STATE BATTING.TEAM NEW.STATE     p.x      p.y    p.EST
##   <chr> <chr>        <chr>       <dbl>    <dbl>    <dbl>
## 1 100 2 SLN          000 2     0.0224   0.0291   0.0291
## 2 100 2 SLN          001 2     0.00673  0.00577  0.00578
## 3 100 2 SLN          010 2     0.0157   0.0220   0.0220
## 4 100 2 SLN          011 2     0.0336   0.0220   0.0221
## 5 100 2 SLN          100 2     0.00448  0.000775 0.000782
## 6 100 2 SLN          101 2     0.0471   0.0435   0.0436
## 7 100 2 SLN          110 2     0.202    0.195    0.196
## 8 100 2 SLN          3         0.668    0.682    0.681
```