Using Monte Carlo Simulation to Predict Baseball Outcomes

Eric Folsom, Eva Halsne, David Tabakian

folsome@wwu.edu, halsnee2@wwu.edu, tabakid@wwu.edu

Math 445, Spring 2022
Instructor: Ramadha Piyadi Gamage

Introduction

We are interested in predicting the number of runs a baseball team scores in a 9-inning baseball game. Baseball is a game that makes heavy use of statistics, and keeping track of player actions made throughout the course of a game is key to understanding the outcomes of a game. In this project, we are exploring the use of Markov Chain Monte Carlo simulations to predict the number of runs a team will score in the course of a game as well as predicting the outcome of a game against an opposing team.

Baseball is a game played between 2 teams consisting of 9 batters, a catcher and a pitcher. Unlike other sports that are played for a set amount of time, baseball is played over the course of 9 innings. This gives each team 9 attempts on offense (batting), and 9 attempts on defense (pitching and protecting bases). Each inning gives each batter 3 chances (strikes) to hit the ball before that batter is considered "out". Each inning is limited to 3 outs, meaning whenever those 3 outs occur, that "side" of the inning is retired, and the teams switch their offensive and defensive positions.

The factors that are taken into account during an "at bat" (the event of a batter appearing at home plate and getting outted) are balls, which is when a pitch is thrown outside of the designated strike box, the strikes mentioned earlier, and the outs. Balls max out at 4, and the batter is allowed a free walk to first base, strikes at 3, and outs at 3. The game progresses as the pitcher pitches to the batters in attempts to end the opposing teams offense. If a player is able to run from the home plate through all three bases and back to home within their side of the inning, that player scores a run. Runs can also be scored by hitting the ball out of the field in the stadium, or completely out of the stadium. The team that scores the most runs in their 9 innings wins the game. In the event of a tie at the end of the 9th inning, extra innings will be played until a winner is decided.

For our project, we synthesized ideas from 3 papers and applied the techniques described in an attempt to predict the outcomes of Major League Baseball (MLB) games using Monte-Carlo simulation and Markov-Chain methods. The three papers that we analyzed are "An Analysis of Baseball Batting Order by Monte Carlo Simulation" by R. Allan Freeze, "How Our MLB Predictions Work" on fivethirtyeight.com, with a model created by Jay Boice, and "RE24" from Fangraphs, written by Neil Weinberg.

The article by Freeze specifically attempts to answer the question on whether batting order on a team influences the outcome of a 9-inning game of baseball. This article lays the groundwork for our research, such as describing a play-by-play method of simulation, as well as implementation of different plays and outcomes in the simulation. In the Weinberg article, we get a better understanding of game states and how we want to describe the state of a game in our simulation. This article discusses the RE24 statistic, or run expectancy (average number of runs an average team would be expected to score during the remainder of the game based on the state of the game) based on the 24 possible game states. These 24 game states are found as a result of having the possibility of zero, one, or two outs, as well as 8 different baserunner arrangements. In our project, we use elements necessary to obtain an RE24 matrix in order to calculate transitional probabilities from one game state to another. "How Our MLB Predictions Work" describes the Monte Carlo simulation methods that align with our goals for this project. However, much of this article describes an Elo-based rating

system, which we did not use, since we are not looking at the performance of each individual player, but the runs scored by the team as a whole.

Our project also utilizes the Markov Chain Monte Carlo methods, as detailed in Chapter 9 of "Statistical Computing with R." The Markov Chain Monte Carlo methods select a stationary pdf $f(\bullet)$, then runs a Markov Chain for a sufficiently long time such that the Markov chain converges to the selected $f(\bullet)$. Most importantly, the Markov Chain allows us to describe a sequence of events where the probability of each event depends only on the previous event. This is useful to our simulation because it allows us to properly sequence the events of a game from start to finish based on the prior outcomes of the game.

Throughout a game, we must be able to describe the current state of events. To do this, we are using a vector of 4 different values that comes from the outcome of a play. The first three values are the number of players on the first, second, and third base respectively, while the fourth value represents the number of outs for the team at bat. For example, the game begins at a (000, 0) state, indicating no runners on first, second, or third, base and no outs. But if the first batter hits a double, the game enters a (010, 0) state, showing that the double has resulted in the batter making it to second base. Additionally, keep in mind that innings are divided into halves, in which the home team and away team are at bat in different halves, so the game state will reset at the end of each half-inning.

Simulation Study

For our simulation study, we simulated a 9 inning game of baseball between two MLB teams using our MCMC model. We wished to see if we could predict the outcome of matchups on a given game by estimating the amount of runs scored for each team. We applied our model to play-by-play data obtained from Retrosheet on the 2021 season. Play-by-play game data was used because the current state of the game and future plays are dependent on what events occur in the previous plays. Specifically, the probability of a future event happening depends on the previous plays, so we need to use play-by-play data in order to estimate the probability of transitioning to a new game state. Markov chains can generally predict an outcome given its current state. Baseball games can be broken down into different game states depending on the number of outs and the position of runners on base using a method as described in the Introduction section. An RE24 matrix highlights this fact by providing the number of runs that are expected to score in the remainder of an inning at each different game state. One of the steps in the process of creating these RE24 matrices is to create a transitional matrix, which tells us the probability that a baseball game transitions from one game state to another. For the purposes of our simulation we must also add one more game state, the state in which there are 3 outs, so that we can end an inning.

In our study, we wrote R code that transformed the play-by-play data from Retrosheet into a transitional matrix for each MLB team. We also had to create a method of determining whether a run scored after a play or not. We are able to determine whether or not a run scored after a play by analyzing the before and after state of a play. The number of runs scored after a play can be calculated by the following formula:

$$\text{Runs} = (R_b + O_b + 1) - (R_a + O_a)$$

Where R_b and O_b are referring to the number of runners on base and number of outs in the current state of the game, and R_a and O_a refer to the number of runners and outs in the new state respectively. We add the $+1$ to the current game state portion of the function to account for the a home run being hit, in which case we must also count the batter as a run that scores.

For example, suppose the state at the beginning of the play had runners on first and third with two outs, meaning the game state is (101, 2). Then, if the batter hits a double, the new game state is (010, 2). We can plug this transition into our formula for calculating runs scored on a play to get:

$$(2 + 2 + 1) - (1 + 2) = 2$$

Here we can see that two runs were scored as the result of the play, which is an accurate result. We can apply this formula to each possible transition to create a runs matrix which tells us how many runs were scored during each different transition.

For another example, we can look at the case of a lead off home run. The starting state for this case is (000,0) and the ending state will also be (000,0). This case illustrates why the $+1$ is needed. To calculate the runs that score as the result of this play in our model we can plug the information that we know into our formula to get,

$$(0 + 0 + 1) - (0 + 0) = 1$$

Telling us that 1 run scored as the result of a lead off home run, which we know to be the case.

Now that we have described a method for calculating the number of runs scored after a play, we can use the transitional matrix that we obtained to simulate a half inning of baseball. For our simulation we always start each inning at the (000, 0) state, with no runners on base and no outs. We simulate an inning batter by batter. To do this, we take a weighted sample from 25 different game states, using the probability of each transition occurring as the weights of the sample. We are taking a random sample of the game states, which gives us the new game state after an at-bat. We then update the state of the game and sample again. This time, the starting state is the updated game state which we found in the prior step. We also reference our Runs matrix to give the number of runs that were scored during the plate appearance. We store the number of runs scored at the end of a plate appearance in a vector. The simulation runs until the end of the half inning, when the our simulation reaches game state 25, 3 outs in an inning.

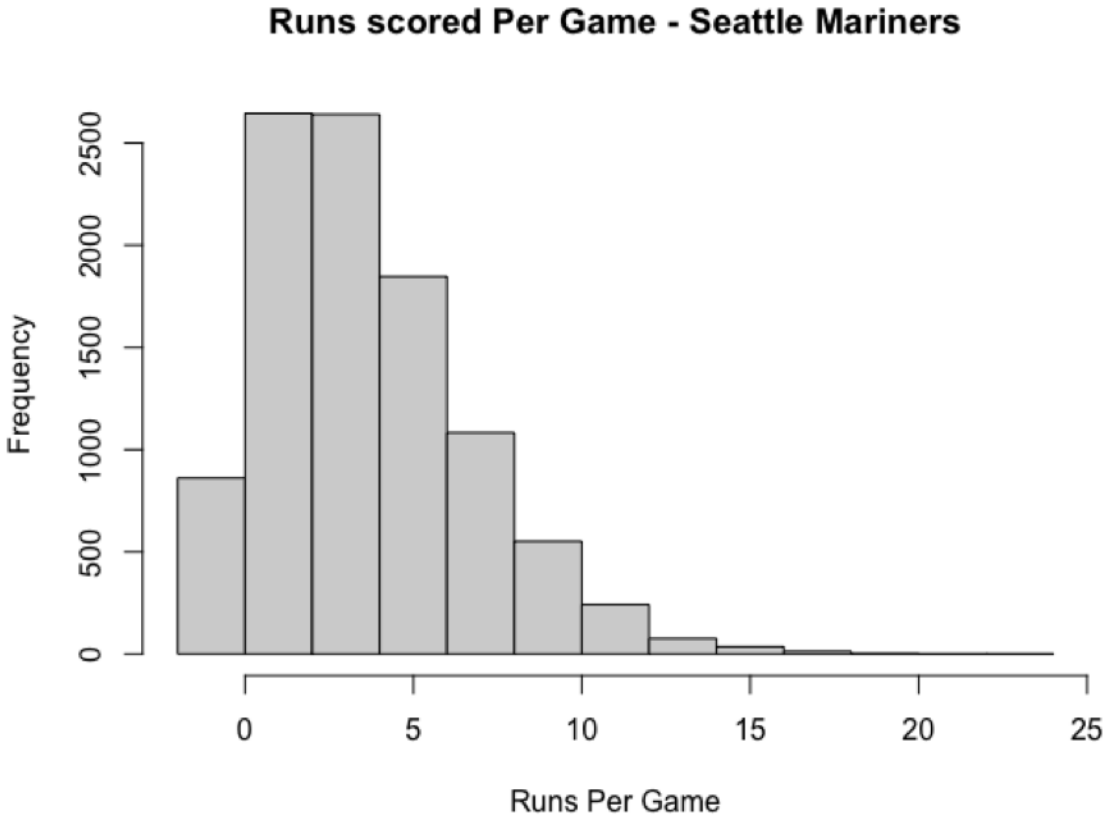
Using our simulation we were able to estimate the number of runs scored per game by a team with MLB average stats. We simulated 10000 games. Our simulation estimate that teams scored on average 4.4107 runs per game in 2021, with a variance of 9.745896 runs per game. The actual MLB average runs per game in 2021 was 4.530671.

To simulate a matchup between two specific teams, we run our simulation using separate transition matrices for each team. Creating the transition matrices for each team is quite straightforward, as we can extract data for each team from the data in Retrosheet. We run our simulation 9 times for each team to simulate a 9 inning game. We then simulated each 9 inning game 10000 times and

obtained the average number of runs that each team scored. We used these averages as our estimate for the outcome of the game.

For a real world application of our model, we can estimate the number of runs scored per game by the Seattle Mariners in 2021. We simulated 10000 games and found the following results.

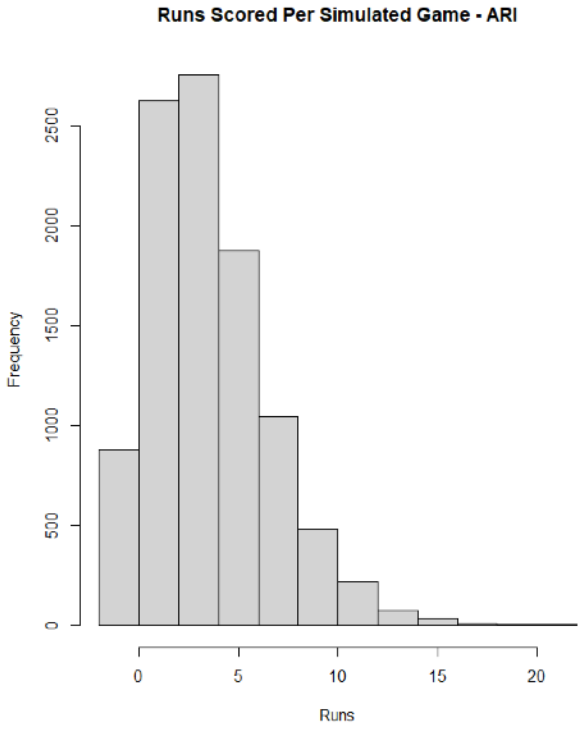
A histogram of the number of runs scored in each simulated game is as follows:

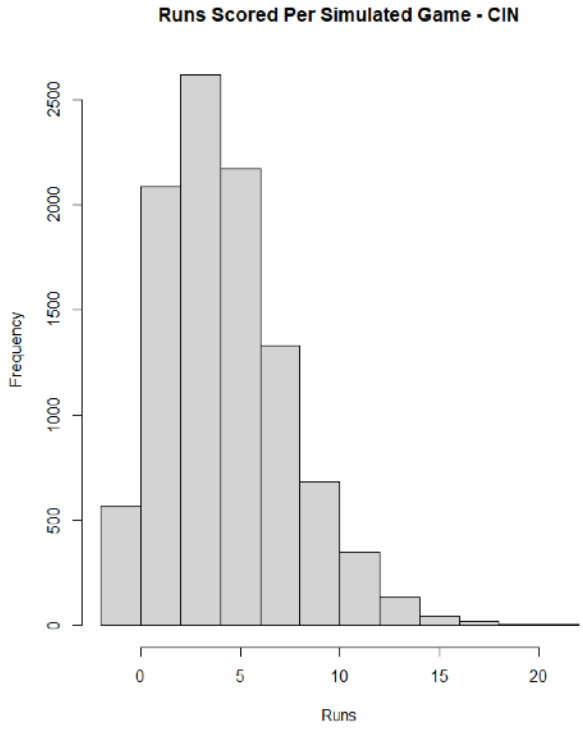


Using our simulation, we obtain an estimated 4.1098 runs per game using data from the 2021 season with a variance of 9.406867. In actuality, the Mariners scored 4.30 runs per game in 2021. We also obtained a 95% percentile based confidence interval for the number of runs scored per game by the Mariners, which was between 0 and 11 runs per game. This confidence interval ends up being accurate to the real-world outcome because there was only one occurrence of the Mariners scoring more than 11 runs during the 2021 season.

For a different real world application of our model, we attempted to predict the results of MLB games on June 6, 2022 using our model. We simulated the following matchups: Arizona vs. Cincinnati, Seattle vs. Houston, Toronto vs. Kansas City, Boston vs. Anaheim, and The New York Mets vs. San Diego.

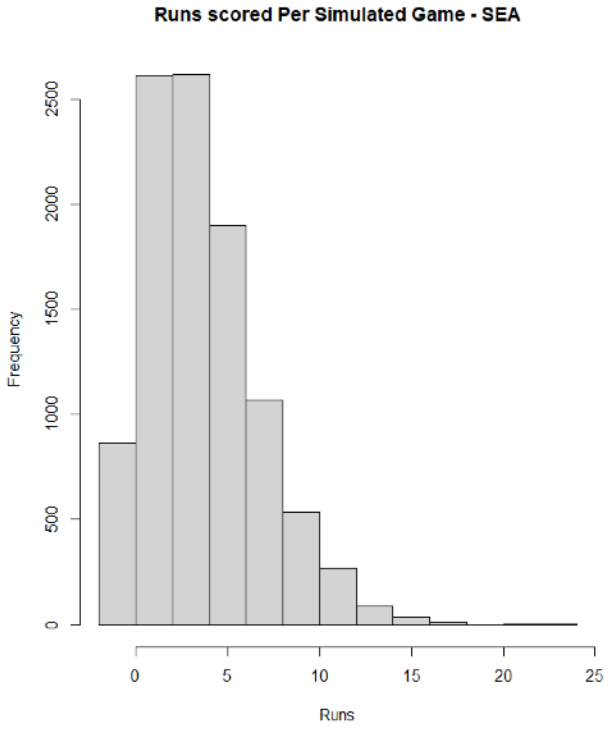
For the game between the Arizona Diamondbacks and the Cincinnati Reds, our simulation predicted that the Reds would win with a score of $5 - 4$. Our simulation produced the following distributions of runs scored for each team.

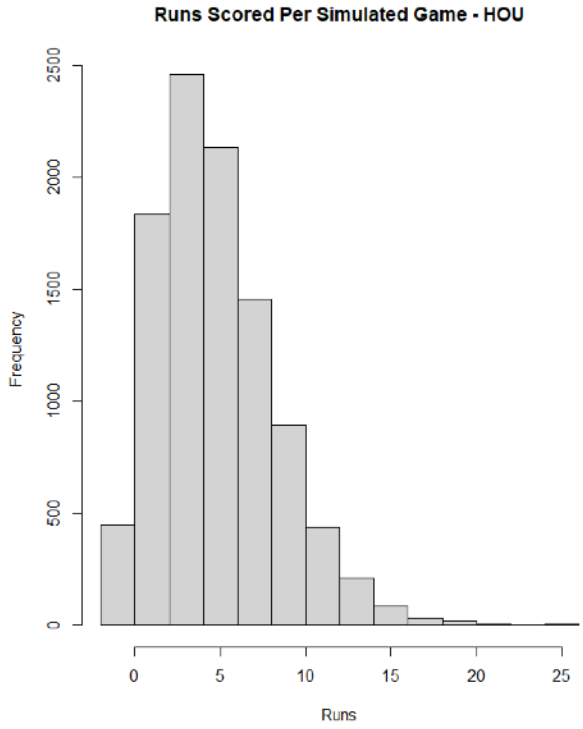




The Reds won this game by a score of $7 - 0$. Our model correctly predicted the winner of the game, but failed to accurately predict the number of runs scored for each team.

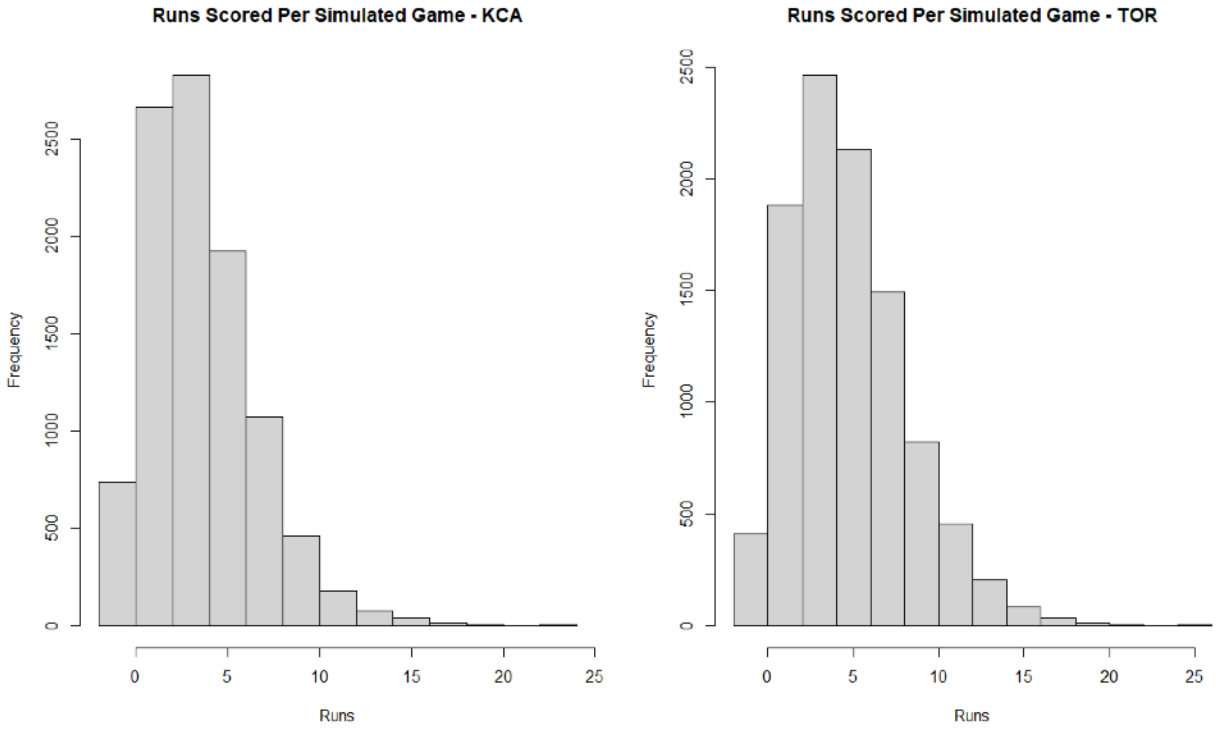
For the game between the Seattle Mariners and the Houston Astros, our model predicted that the Astros would win with a score of $5 - 4$. Our simulation produced the following distributions of runs scored for each team.





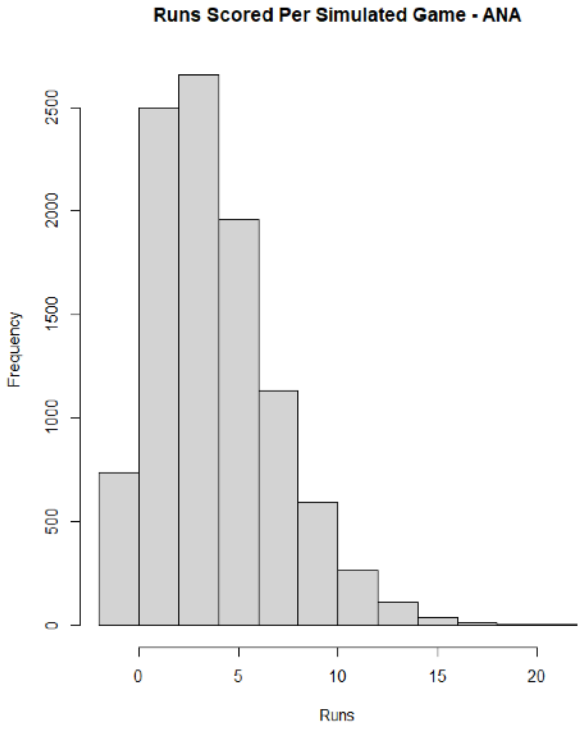
The Mariners won this game by a score of $7-4$. Our model did not accurately predict the winner or the number of runs scored in this game.

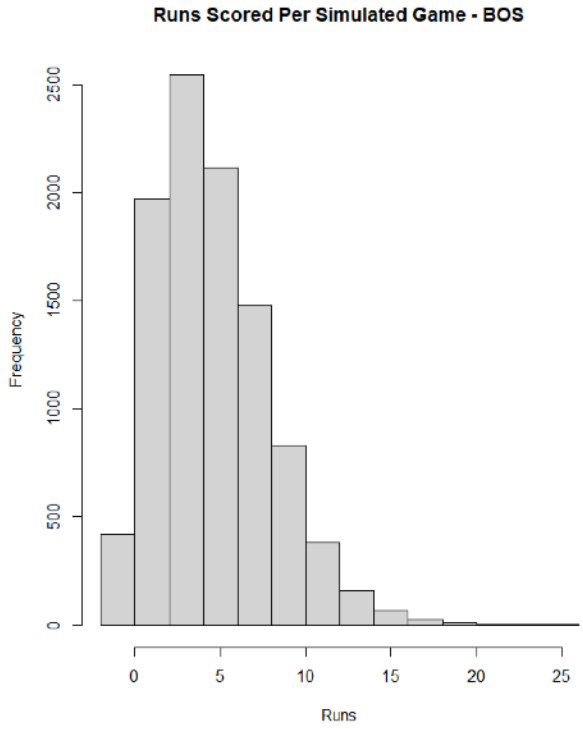
For the game between the Toronto Blue Jays and the Kansas City Royals our simulation predicted that the Blue Jays would win with a score of $6-5$. Our simulation produced the following distributions of runs scored for each team.



The Blue Jays won this game by a score of $8-0$. Our model correctly predicted the winner of this game, but it was not accurate on the number of runs scored by each team.

For the game between the Boston Red Sox and the Los Angeles Angles of Anaheim our simulation predicted that the Red Sox would win with a score of $6-5$. Our simulation produced the following distributions of runs scored by each team.

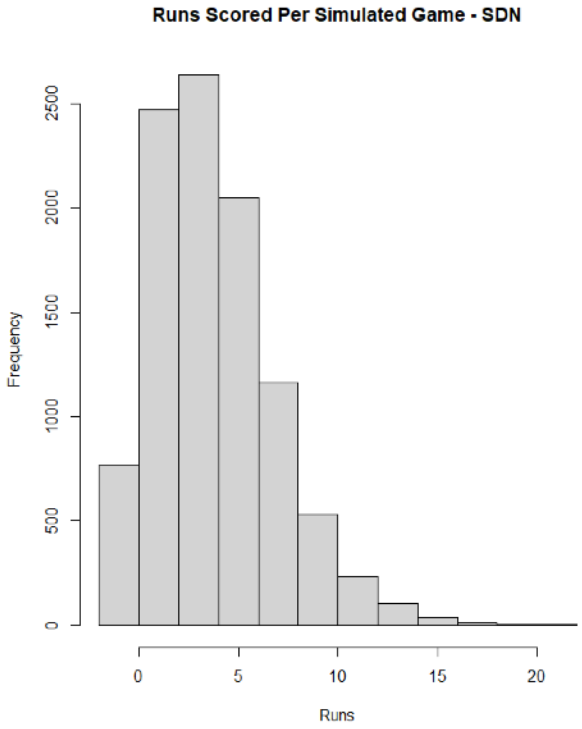


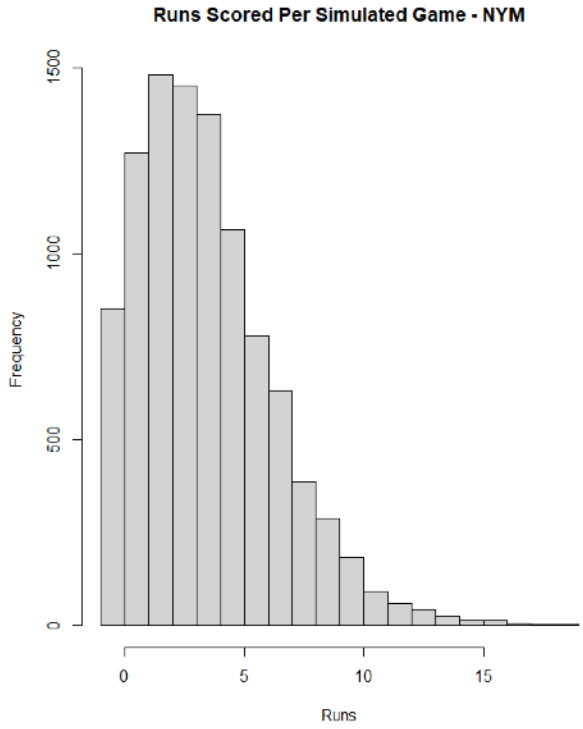


Boston won this game by a score of $1-0$. Our model correctly predicted the winner of the game, but it failed to predict the number of runs scored by each team.

For the game between the New York Mets and the San Diego Padres our simulation predicted that the Padres would win with a score of $5-4$. Our simulation produced the following distributions of runs scored by each team.

The Mets won this game by a a score of $11-5$. Our model failed to predict the winner of the game and failed to accurately predict the number of runs each team scored.





Our model had mostly poor results when it comes to simulating specific matchups. Our model successfully predicted the winner of 3 out of the 5 outcomes, but it was not able to accurately predict the number of runs that each team scored in any of the games. Therefore, we think it is purely coincidental that the simulation correctly predicted the winner in these cases.

Discussion

From the results of our simulation, we can see that there are some clear limitations to our model. For one, we are attempting to use our model to predict outcomes during the 2022 season. This is a problem because the data that we are using is taken from the 2021 season. Due to year to year changes in team composition and individual player performance, the data that we have for each team will not be an accurate representation of the team's ability from season to season. There are a few different ways that we could go about remedying this issue. One remedy would be to take data from the same season that we are trying to predict outcomes on. The issue with this remedy is the complexity involved with transforming up-to-date data into usable data for our purposes. Another possible remedy to this issue would be to compute the transitional matrix on a player by player basis. Obtaining a player-specific transitional matrix would allow us to create a lineup for a given night, which would create more accurate simulations for each at-bat, because right now our simulation assumes that each batter on the team performs to the team average. We believe that if we were to implement one of these two remedies we would be able to more accurately predict the outcomes of specific matchups.

As noted from our real world simulations, our model does a fairly good job at estimating the number of runs that each team scores per game. For further study, we can use the same idea for predicting the number of runs scored in a game to predict the number of runs that a team allows per game. From here we can use the Pythagorean expectation derived by Bill James to estimate the

winning percentage of a team in a given season. Furthermore, being able to estimate the number of runs that a team allows per game will allow us to make better predictions as to the outcomes of the game. We could possibly look into whether or not there is a method of precisely estimating how many runs a team will score based on how many runs their opponent gives up on average.

When we simulated our matchups, we noticed that the outcome of each game was decided by 1 run. We believe this to be the case that because we are only simulating the batter outcomes for each team. If we were to include simulations for the number of runs that a team allows per game, we believe that we could obtain a more accurate prediction for the score of each MLB match-up.

The process above is the next step in strengthening this model, by taking the ratio of runs scored over runs allowed, we can get a better idea of a teams win probability and use that to calculate win percentage. We can simulate runs allowed in a similar manner to runs scored and have a better estimate of win percentage. If we were to take this study farther, we would take this better win percentage and use that to better predict games and possibly simulate an entire season.

Appendix

Bibliography:

Boice, Jay. "HOW OUR MLB Predictions Work." FiveThirtyEight, FiveThirtyEight, 28 Mar. 2018, https://fivethirtyeight.com/methodology/how-our-mlb-predictions-work/.

Freeze, R Allen. "An Analysis of Baseball Batting Order by Monte Carlo Simulation - JSTOR." Jstor.org, 25 June 1973, https://www.jstor.org/stable/169949.

Weinberg, Neil. "RE24." Sabermetrics Library, 30 July 2014, https://library.fangraphs.com/misc/re24/.

Play-by-play data was pulled from: https://www.retrosheet.org/game.htm

R Code:

```
###Math 445 Final Project R Code - Eric Folsom, Eva Halsne, David Tabakian##
library(plyr)
library(dplyr)
library(readr)
library(car)
###In R studio set working directory to Source File Location###
data2021 <- read_csv("all2021.csv", ##initializing play by play data from
    col_names = FALSE)#retrosheet for the 2021 season
fields <- read.csv("fields.csv")#Play by play data does not come wiith the headers we want
names(data2021) <- fields[, "Header"]#adding the column names that we want.

data2021$HALF.INNING <- with(data2021, #tells us what half inning the game is in
    paste(GAME_ID, INN_CT, BAT_HOME_ID))#top or bottom
data2021$RUNS.SCORED <- with(data2021, (BAT_DEST_ID > 3) + #how many runs score as the resul
    (RUN1_DEST_ID > 3) + (RUN2_DEST_ID > 3) + (RUN3_DEST_ID > 3))
RUNS.SCORED.INNING <- aggregate(data2021$RUNS.SCORED,#count number of runs scored in an inni
     list(HALF.INNING=data2021$HALF.INNING), sum)
RUNS.SCORED.START <- aggregate(data2021$RUNS,#runs scored at start of an inning
    list(HALF.INNING=data2021$HALF.INNING), "[", 1)
#maximum number of runs scored in an inning
MAX <- data.frame(HALF.INNING=RUNS.SCORED.START$HALF.INNING)
MAX$x <- RUNS.SCORED.INNING$x + RUNS.SCORED.START$x
data2021 <- merge(data2021, MAX)
N <- ncol(data2021)
names(data2021)[N] <- "MAX.RUNS"
data2021$RUNS.ROI <- with(data2021, MAX.RUNS - RUNS)
##This function returns the state of the game based on the position or runners
get.state <- function(runner1, runner2, runner3, outs){#and the number of outs
  runners <- paste(runner1, runner2, runner3, sep="")
  paste(runners, outs)
}
```

```
##getting runner positions from the data into a readable form for our function
RUNNER1 <- ifelse(as.character(data2021[ ,"BASE1_RUN_ID"]) == "", 0, 1)#get.state
RUNNER1[is.na(RUNNER1)] <- 0
RUNNER2 <- ifelse(as.character(data2021[ ,"BASE2_RUN_ID"]) == "", 0, 1)
RUNNER2[is.na(RUNNER2)] <- 0
RUNNER3 <- ifelse(as.character(data2021[ ,"BASE3_RUN_ID"]) == "", 0, 1)
RUNNER3[is.na(RUNNER3)] <- 0

##Obtaining the game states at the beginning and end of the plays
data2021$STATE <- get.state(RUNNER1, RUNNER2, RUNNER3, data2021$OUTS_CT)
NRUNNER1 <- with(data2021, as.numeric(RUN1_DEST_ID == 1 |
        BAT_DEST_ID == 1))
NRUNNER2 <- with(data2021, as.numeric(RUN1_DEST_ID == 2 |
             RUN2_DEST_ID == 2 | BAT_DEST_ID==2))
NRUNNER3 <- with(data2021, as.numeric(RUN1_DEST_ID == 3 |
             RUN2_DEST_ID == 3 | RUN3_DEST_ID == 3 | BAT_DEST_ID == 3))
NOUTS <- with(data2021, OUTS_CT + EVENT_OUTS_CT)
data2021$NEW.STATE <- get.state(NRUNNER1, NRUNNER2, NRUNNER3, NOUTS)

#creating a new data frame that only looks that the the ends of the innings
#this is so we can look at the states where there are 3 outs in an inning.
data2021 <- subset(data2021, (STATE != NEW.STATE) | (RUNS.SCORED > 0))
data.outs <- ddply(data2021, .(HALF.INNING), summarize,
                   Outs.Inning=sum(EVENT_OUTS_CT))
data2021 <- merge(data2021, data.outs)
data2021C <- subset(data2021, Outs.Inning == 3)
data2021C <- subset(data2021, BAT_EVENT_FL == TRUE)
data2021C$NEW.STATE <- recode(data2021C$NEW.STATE,# recoding all the sates with 3 outs
"c('000 3', '100 3', '010 3', '001 3','110 3', '101 3', '011 3', '111 3')='3'")# to state 3

RUNS <- with(data2021C, aggregate(RUNS.ROI, list(STATE), mean))
RUNS$Outs <- substr(RUNS$Group, 5, 5)
RUNS <- RUNS[order(RUNS$Outs), ]
RUNS.2021 <-matrix(round(RUNS$x, 2), 8, 3)
dimnames(RUNS.2021)[[2]] <- c("0 outs", "1 out", "2 outs")
dimnames(RUNS.2021)[[1]] <- c("000", "001", "010", "011", "100", "101",
                              "110", "111")

#Constructing our transitional matrix
T.matrix <- with(data2021C, table(STATE, NEW.STATE))
```

```
P.matrix <- prop.table(T.matrix, 1)
P.matrix <- rbind(P.matrix, c(rep(0, 24), 1))
#our new transitional matrix with 25 game states

#Creating a function taht counts the number of runners and outs in a game state
count.runners.outs <- function(s)
  sum(as.numeric(strsplit(s,"")[[1]]), na.rm=TRUE)

#creating a runs matrix so we can see how many runs score as the result of the play
runners.outs <- sapply(dimnames(T.matrix)[[1]], count.runners.outs)[-25]
R <- outer(runners.outs + 1, runners.outs, FUN="-")
dimnames(R)[[1]] <- dimnames(T.matrix)[[1]][-25]
dimnames(R)[[2]] <- dimnames(T.matrix)[[1]][-25]
R <- cbind(R, rep(0, 24))#runs matrix

##This is our function that simulates the half inning.
#inputs are the transitional matrix, runs matrix, and the starting state
#The function returns the number of runs scored in the simulated half inning
simulate.half.inning <- function(P, R, start=1){
  s <- start; path <- NULL; runs <- 0
  while(s < 25){
    s.new <- sample(1:25, 1, prob=P[s, ])
    path <- c(path, s.new)
    runs <- runs + R[s, s.new]
    s <- s.new
  }
  runs
}


##Simulating 10000 games of baseball by the average MLB team in 2021##
G<-10000
game.sim<-double(G)
for (i in 1:G){
  game.sim[i]<-sum(replicate(9,simulate.half.inning(T.matrix,R)))
}
mean(game.sim)
var(game.sim)
hist(game.sim)
```

```
#Obtaining transitional matricies for each MLB team based on 2021 data.
data2021C$HOME_TEAM_ID <- with(data2021C, substr(GAME_ID, 1, 3))
data2021C$BATTING.TEAM <- with(data2021C,
                               ifelse(BAT_HOME_ID == 0,
                                      as.character(AWAY_TEAM_ID),
                                      as.character(HOME_TEAM_ID)))
Team.T <- with(data2021C, table(BATTING.TEAM, STATE, NEW.STATE))
ANA.t<-Team.T['ANA', , ]
ARI.t<-Team.T['ARI', , ]
ATL.t<-Team.T['ATL', , ]
BAL.t<-Team.T['BAL', , ]
BOS.t<-Team.T['BOS', , ]
CHA.t<-Team.T['CHA', , ]
CHN.t<-Team.T['CHN', , ]
CIN.t<-Team.T['CIN', , ]
CLE.t<-Team.T['CLE', , ]
COL.t<-Team.T['COL', , ]
DET.t<-Team.T['DET', , ]
HOU.t<-Team.T['HOU', , ]
KCA.t<-Team.T['KCA', , ]
LAN.t<-Team.T['LAN', , ]
MIA.t<-Team.T['MIA', , ]
MIL.t<-Team.T['MIL', , ]
MIN.t<-Team.T['MIN', , ]
NYA.t<-Team.T['NYA', , ]
NYM.t<-Team.T['NYN', , ]
OAK.t<-Team.T['OAK', , ]
PHI.t<-Team.T['PHI', , ]
PIT.t<-Team.T['PIT', , ]
SDN.t<-Team.T['SDN', , ]
SEA.t<-Team.T['SEA', , ]
SFN.t<-Team.T['SFN', , ]
SLN.t<-Team.T['SLN', , ]
TBA.t<-Team.T['TBA', , ]
TEX.t<-Team.T['TEX', , ]
TOR.t<-Team.T['TOR', , ]
WAS.t<-Team.T['WAS', , ]

########################################
######6/6/2022 game outcome prediction###
########################################
```

```
B<-10000
par(mfrow=c(1,2))
#############
##ARI @ CIN##
#############

ARI.score<-double(B)
CIN.score<-double(B)
runs.ARI<-double(18)
runs.CIN<-double(18)
for(i in 1:B){

  for( j in 1:18){
    if(j %%2==0)
      runs.CIN[j]<-simulate.half.inning(CIN.t,R)

    else
      runs.ARI[j]<-simulate.half.inning(ARI.t,R)
  }
  ARI.score[i]<-sum(runs.ARI)
  CIN.score[i]<-sum(runs.CIN)
}
mean(CIN.score)
mean(ARI.score)

hist(ARI.score, main="Runs Scored Per Simulated Game - ARI", xlab="Runs")
hist(CIN.score, main="Runs Scored Per Simulated Game - CIN", xlab="Runs")

#Our simulation predicts that Cincinatti will beat Arizona with a score of 5-4
#Actual outcome: CIN 7 - ARI 0
#correct winner

#############
##TEX @ CLE##
#############

TEX.score<-double(B)
CLE.score<-double(B)
runs.TEX<-double(18)
runs.CLE<-double(18)
```

```
for(i in 1:B){

  for( j in 1:18){
    if(j %%2==0)
      runs.CLE[j]<-simulate.half.inning(CLE.t,R)

    else
      runs.TEX[j]<-simulate.half.inning(TEX.t,R)
  }
  TEX.score[i]<-sum(runs.TEX)
  CLE.score[i]<-sum(runs.CLE)
}
mean(CLE.score)
mean(TEX.score)

hist(CLE.score, main="Runs Scored Per Simulated Game - CLE", xlab="Runs")
hist(TEX.score, main="Runs Scored Per Simulated Game - TEX", xlab="Runs")

#Our simulation predicts that Cleveland will beat Texas with a score of 5-4
#Actual Outcome: Game Postponed
#N/A

##############
##TOR @ KCA##
##############

TOR.score<-double(B)
KCA.score<-double(B)
runs.TOR<-double(18)
runs.KCA<-double(18)
for(i in 1:B){

  for( j in 1:18){
    if(j %%2==0)
      runs.KCA[j]<-simulate.half.inning(KCA.t,R)

    else
      runs.TOR[j]<-simulate.half.inning(TOR.t,R)
  }
  TOR.score[i]<-sum(runs.TOR)
  KCA.score[i]<-sum(runs.KCA)
```

```
}

mean(KCA.score)
mean(TOR.score)

hist(KCA.score, main="Runs Scored Per Simulated Game - KCA", xlab="Runs")
hist(TOR.score, main="Runs Scored Per Simulated Game - TOR", xlab="Runs")

#Our simulation predicts that Toronto will beat Kansas City  by a score of 5-4
#Actual Outcome: TOR 8 - KCA 0
#correct winner

#############
##BOS @ ANA##
#############

BOS.score<-double(B)
ANA.score<-double(B)
runs.BOS<-double(18)
runs.ANA<-double(18)
for(i in 1:B){

  for( j in 1:18){
    if(j %%2==0)
      runs.ANA[j]<-simulate.half.inning(ANA.t,R)

    else
      runs.BOS[j]<-simulate.half.inning(BOS.t,R)
  }
  BOS.score[i]<-sum(runs.BOS)
  ANA.score[i]<-sum(runs.ANA)
}
mean(ANA.score)
mean(BOS.score)

hist(ANA.score, main="Runs Scored Per Simulated Game - ANA", xlab="Runs")
hist(BOS.score, main="Runs Scored Per Simulated Game - BOS", xlab="Runs")

#Our simulation predicts that Boston will beat Anaheim with a score of 5-4
#Actual outcome: BOS 1 - ANA 0
#correct winner
```

```
#############
##NYM @ SDN##
#############

NYM.score<-double(B)
SDN.score<-double(B)
runs.NYM<-double(18)
runs.SDN<-double(18)
for(i in 1:B){

  for( j in 1:18){
    if(j %%2==0)
      runs.SDN[j]<-simulate.half.inning(SDN.t,R)

    else
      runs.NYM[j]<-simulate.half.inning(NYM.t,R)
  }
  NYM.score[i]<-sum(runs.NYM)
  SDN.score[i]<-sum(runs.SDN)
}
mean(SDN.score)
mean(NYM.score)

hist(SDN.score, main="Runs Scored Per Simulated Game - SDN", xlab="Runs")
hist(NYM.score, main="Runs Scored Per Simulated Game - NYM", xlab="Runs")

#Our simulation predicts that San Diego will beat New York (N) with a score of 5-4
#Actual Outcome: NYM 11 - SDN 5
#incorrect

#############
##SEA @ HOU##
#############

SEA.score<-double(B)
HOU.score<-double(B)
runs.SEA<-double(18)
runs.HOU<-double(18)
for(i in 1:B){
```

```
  for( j in 1:18){
    if(j %%2==0)
      runs.HOU[j]<-simulate.half.inning(HOU.t,R)

    else
      runs.SEA[j]<-simulate.half.inning(SEA.t,R)
  }
  SEA.score[i]<-sum(runs.SEA)
  HOU.score[i]<-sum(runs.HOU)
}
mean(HOU.score)
mean(SEA.score)

hist(SEA.score, main = "Runs scored Per Simulated Game - SEA",xlab="Runs" )
hist(HOU.score, main="Runs Scored Per Simulated Game - HOU", xlab="Runs")
mean(SEA.score)
var(SEA.score)
quantile(SEA.score,0.025)
quantile(SEA.score,0.975)

#Our simulation predicts that Houston will beat Seattle with a score of 6-5
#Actual Results: SEA 7 - HOU 4
#Incorrect
```