Math 23A R term project

Maggie Ji and Emmet Murphy

12/7/2017

War Simulations: From Card Game to Korea

Overview

The first part of this paper generates a Markov process to predict the probability of a hand winning in the War card game.

The second part turns toward a more serious application: Using Bayesian updating to model the outcomes of a real war, such as with North Korea.

Generating a Markov Process

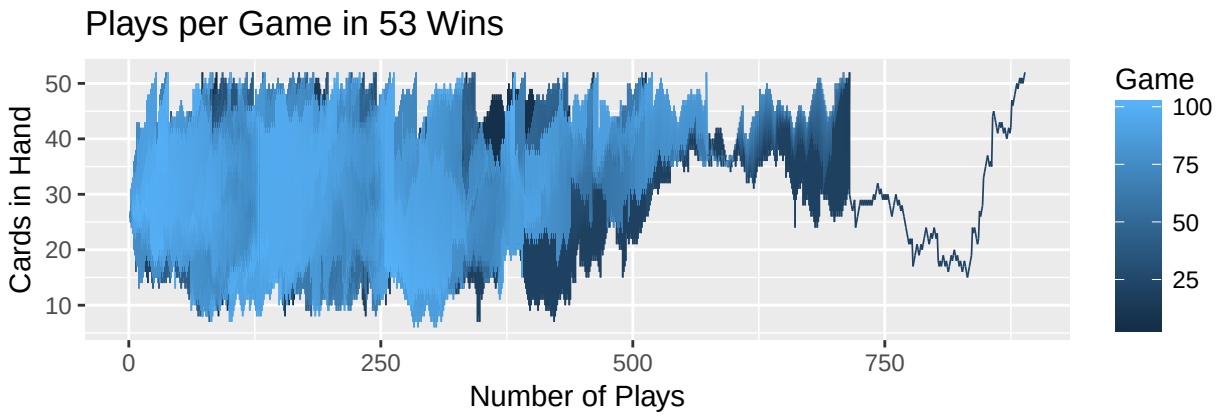
Our goal is to predict the probability of a hand winning in n number of plays, given x number of cards in the hand. We accomplish it in three steps:

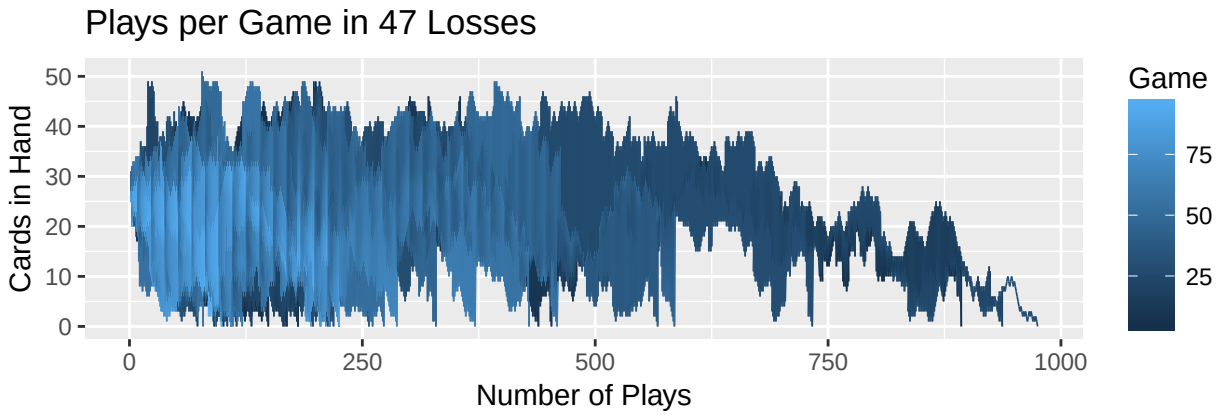
1. Simulate games, collect and explore the data
2. Model the probabilities as a Markov Chain
3. Predict a hand's probability of winning

Part 1: Simulate and Explore the Data

In order to train our Markov process, we first simulate 100 games of War (see the Appendix for details). The simulations generate and store data on starting hand strength, the number of cards in a player's hand after each play, etc. The following script reads in the data and plots the number of cards by the number of plays. The plots give a sense of the dispersion of outcomes.

```
# plot number of cards in hand over the course of games
deck_sizes <- read_csv("deckSizes100random.csv")
games <- read_csv("no-ties/games100random.csv")
gamePlays = merge(deck_sizes, games, by = 'id')
gamePlays$cards = gamePlays$p1Cards
gamePlays$Game = gamePlays$id
wins <- gamePlays[gamePlays$result=='W',]
losses <- gamePlays[gamePlays$result=='L',]
ties <- gamePlays[gamePlays$result=='T',]
p1 <- ggplot(data=wins, aes(x=play, y=cards)) + geom_line(aes(color=Game), size=0.3) +
    ggtitle(paste("Plays per Game in", length(unique(wins[,'id'])), "Wins")) +
    xlab('Number of Plays') + ylab("Cards in Hand")
p2 <- ggplot(data=losses, aes(x=play, y=cards)) + geom_line(aes(color=Game), size=0.3) +
    ggtitle(paste("Plays per Game in", length(unique(losses[,'id'])), "Losses")) +
    xlab('Number of Plays') + ylab("Cards in Hand")
grid.arrange(p1, p2, ncol=1)
```





To a player it can seem like a game of War will go on forever (as discussed in more detail in the Appendix). The following summarizes how many plays our simulated games require in order to converge on a winner:

```
summary(games$plays)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    26.0   121.5   211.0   275.7   372.5   974.0
```

Part 2: Generate a Markov Process

The Markov chain calls for a 53 x 53 matrix M (there are 52 cards and a +1 for the possibility of a hand ending with 0 cards). Each cell in row i, column j has a probability of a hand going from $i-1$ number of cards to $j-1$ number of cards.

The probability of a hand with x cards winning in n number of plays is then:

$$M^n \vec{e_x}$$

Following is the script to generate the Markov matrix M:

```
# generate Markov matrix
# for probabilities, this is the denominator -- frequency of each hand size
handCountFreq <- plyr::count(gamePlays, vars='cards')

# the numerator is trickier -- need to count how many times a hand size
# follows any given hand size
gamePlays$nextCards <- lead(gamePlays$cards, 1)
counts <- ddply(gamePlays, .(gamePlays$cards, gamePlays$nextCards))
```

```
names(counts) <- c("cards", "nextCards", "Freq")
# The probability of transitioning from column p to column q
probTransit <- function(p, q) {
    # Columns p, q correspond to card counts p-1, q-1
    x <- p - 1; y <- q -1
    numerator <- counts[counts$cards == x & counts$nextCards == y,]
    denominator <- handCountFreq[handCountFreq$cards == x,]$freq
    prob <- (nrow(numerator) / denominator)
    if (is.na(prob) || is.nan(prob)) {
        prob <- 0
    }
    return (prob)
}

M <- matrix(0L, nrow=53, ncol=53)
# Fill the transitions to 0 and 52 cards outside the main loop, since the
# transitions are only one-way
for (j in 2:52) {
    M[1,j] <- probTransit(j, 1)
    M[53,j] <- probTransit(j, 53)
}
# When have 0 or 52 cards, probability of keeping that many is 1
M[1,1] = 1.0
M[53,53] = 1.0

# populate the matrix
# value in cell i,j is the likelihood of moving from i to j
for (i in 2:52) {
    for (j in 2:52) {
        if (i == j) {
            # we resolve ties within a play, so the number of cards always changes
            # between plays (i.e. probability of staying on a card count is 0)
            M[i,j] <- 0.0
        } else {
            M[i,j] <- probTransit(i, j)
        }
    }
}
```

Part 3: Prediction

Generating the Markov matrix is the difficult part. Predicting a hand's probability of winning then simply calls for a function to perform the matrix multiplication $M^n \vec{e_x}$.

```
# predict the probability of a hand with x number of cards winning after n plays
winProbability <- function(M, numCards, numPlays) {
    # create a vector for the starting position
    v <- numeric(53)
    v[numCards + 1] = 1

    # multiply by the Markov Matrix to get the probability of winning
    for (i in 1:numPlays) {
        M <- M %*% M
```

```
    }
    Mprob <- M %*% v
    return (Mprob[53])
}
data.frame(OneIn10 = winProbability(M, 1,10), FiftyOneInNext=winProbability(M,51,1),
           TwentySixIn20=winProbability(M,26,20))
```

```
##      OneIn10 FiftyOneInNext TwentySixIn20
## 1 0.02061325      0.4886364     0.2449805
```

The sample values above show the probability of starting with:

- 1 card and winning in 10 plays
- 51 cards and winning on the very next play
- 26 cards and winning in the next 20 plays

Further research would be to incorporate hand strength when calculating the probability, a non-trivial improvement, as it calls for extending the model to include conditional probabilities.

Real World Application: Bayesian Updating in a State of War

So far, our simulations of the card game have assumed that the deck held by each player is randomly drawn, so that neither player knows (1) whether the decks are unequal; and (2) who has the better deck. This begs the question: is it possible to use the information provided by the already revealed cards to make inferences about the strength of a player's deck? *POINT: Includes two related but distinct topics.*

Indeed, in real-world situations that are analogous to this simple card game, a key first step is to assess the position of your opponent, relative to your own position. *POINT: Project is related to a topic that you have studied in another course this term: API-302 Analytic Frameworks for Policy.* Take the case of the U.S. and North Korea as an example. In the face of escalating tensions, each side is desperately trying to figure out what "deck of cards" the other holds, in terms of nuclear capabilities. Although there is not officially a state of war, there are sufficiently many small-scale movements by each side, through which information can be collected. Repeated interactions between the U.S. and North Korea, in the form of diplomatic relations, missile testing, and military demonstrations, could reveal much about each country's true capabilities.

For ease of demonstration, we will make the simplifying assumption that there is no bluffing or strategic omission of information involved on either side (which is admittedly unrealistic). Therefore, the outcome of each interaction is purely based on a random draw from each country's true distribution of military strength (i.e. their "deck of cards").

How Bayesian Updating Works

To start, let's assume that the U.S. holds an infinite arsenal of weapons ranging in power from 1 to 10. North Korea could either have a stronger arsenal, of weapons ranging between 6 and 10 in power, or a weaker arsenal, of weapons ranging between 1 and 5. Through repeated interactions in the form of small interactions, the U.S. can use Bayesian updating to inform its perception of the true range of North Korea's nuclear arsenal.

R CODE: CONDITIONAL PROBABILITIES

As shown by these permutation tests:

(A) If North Korea has a relatively weaker arsenal, then the U.S. would win 70 percent of the small-scale conflicts, lose 20 percent of the time, and tie with North Korea during the remaining 10 percent of conflicts: $p(win|weaker) = 0.7, p(lose|weaker) = 0.2, p(tie|weaker) = 0.1$

(B) However, if North Korea has a relatively stronger arsenal, then the probabilities are reversed: $p(win|stronger) = 0.2, p(lose|stronger) = 0.7, p(tie|stronger) = 0.1$

Say that the U.S. starts with no knowledge of North Korea's true strength, such that it is equally possible that it has a stronger arsenal or a weaker arsenal. In other words, the U.S. holds the prior that $p(weaker) = 0.5$ and $p(stronger) = 1 - p(weaker) = 0.5$. (Note here that we can focus on only $p(weaker)$ without loss of information, since $p(stronger) = 1 - p(weaker)$.)

If a small military conflict erupts in the DMZ between U.S. troops and North Korean forces, then the U.S. can update its perception of North Korea's likely strength, based on the outcome of that conflict: (1) If the U.S. wins, then $p(weaker|win)$ becomes the new prior, rather than 0.5: $p(weaker|win) = \frac{p(win|weaker)p(weaker)}{p(win)}$ $= \frac{p(win|weaker)p(weaker)}{p(win|weaker)p(weaker)+p(win|stronger)p(stronger)} = \frac{0.7*0.5}{0.7*0.5+0.2*0.5}$

(2) If the U.S. loses, then $p(weaker|lose)$ becomes the new prior, rather than 0.5: $p(weaker|lose) = \frac{p(lose|weaker)p(weaker)}{p(lose)} = \frac{p(lose|weaker)p(weaker)}{p(lose|weaker)p(weaker)+p(lose|stronger)p(stronger)} = \frac{0.2*0.5}{0.2*0.5+0.7*0.5}$

(3) If there is a tie, then $p(weaker|tie)$ becomes the new prior (in this particular case, it is still 0.5): $p(weaker|tie) = \frac{p(tie|weaker)p(weaker)}{p(tie)} = \frac{p(tie|weaker)p(weaker)}{p(tie|weaker)p(weaker)+p(tie|stronger)p(stronger)} = \frac{0.1*0.5}{0.1*0.5+0.1*0.5}$

To generalize, we can define a sequence of updated priors as: $P = \{p_1, p_2, p_3, ...\} = \{p_n | n \in \mathbb{R}\}$ where n denotes the nth small-scale conflict. This is a somewhat complicated sequence, where $p_n = f(p_{n-1})$ is a different function for each possible outcome: $p_n = \frac{0.7p_{n-1}}{0.7p_{n-1}+0.2(1-p_{n-1})}$ if the U.S. wins in the nth conflict; $p_n = \frac{0.2p_{n-1}}{0.2p_{n-1}+0.7(1-p_{n-1})}$ if the U.S. loses in the nth conflict; and $p_n = \frac{0.1p_{n-1}}{0.1p_{n-1}+0.1(1-p_{n-1})}$ if there is a tie in the nth conflict.

R CODE: SINGLE SIMULATION OF BAYESIAN UPDATING

In this example, a sequence of randomly drawn outcomes is simulated based on underlying "true" distributions where the U.S. has a stronger arsenal than North Korea. Here, the sequence of updated Bayesian probabilities are as follows:

$$p_0 = 0.0000000$$
$$p_1 = 0.7777778$$
$$p_2 = 0.9245283$$
$$p_3 = 0.9772080$$
$$p_4 = 0.9933802$$
$$p_5 = 0.9980996$$
$$\dots$$
$$p_{14} = 1.0000000$$
$$\dots$$

Convergence to Underlying Distribution

In the previous simulation, the sequence P of Bayesian updated probabilities converges to 1 around the 14th trial, reflecting the true underlying distribution (i.e. that the U.S. has a stronger arsenal). Is this always the case?

Let us represent the process of Bayesian updating using terminology from Math 23A. *POINT: Incorporates ideas both from linear algebra and from real analysis.* The updating function depends on the outcome of each interaction, such that it can be represented as:

$$p_n = [\vec{\mathbf{B}}]\vec{v_n}(p_{n-1})$$

where $\vec{\mathbf{B}} = \begin{bmatrix} f_{win} & f_{lose} & f_{tie} \end{bmatrix} = \begin{bmatrix} \frac{0.7*p}{0.7*p+0.2*(1-p)} & \frac{0.2*p}{0.2*p+0.7*(1-p)} & \frac{0.1*p}{0.1*p+0.1*(1-p)} \end{bmatrix}$ and $\vec{v_n} = \begin{bmatrix} win \\ lose \\ tie \end{bmatrix}$ represents the outcome of each interaction.

For example, the outcome where the U.S. wins would be represented as $\vec{v_n} = \begin{bmatrix}1\\0\\0\end{bmatrix}$, with the updating function $[\vec{\mathbf{B}}]\vec{v_n}(p_{n-1}) = \begin{bmatrix}f_{win} & f_{lose} & f_{tie}\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix} = f_{win} = \frac{0.7*p_{n-1}}{0.7*p_{n-1}+0.2*(1-p_{n-1})}$.

In general, after k number of interactions, the updated prior would be some function of the initial starting prior p_0, depending on how the sequence of outcomes turns out from the interactions:

$$p_k = [\vec{\mathbf{B}}]\vec{v_k}[\vec{\mathbf{B}}]\vec{v_{k-1}}\dots[\vec{\mathbf{B}}]\vec{v_1}(p_0)$$

There are two useful properties of this sequence, that make Bayesian updating useful:

(A) The marginal value of information diminishes with each interaction, such that there is an optimal stopping point after a finite number of interactions:

$$\lim_{n\to\infty} p_n - p_{n-1} = \lim_{n\to\infty}([\vec{\mathbf{B}}]\vec{v_n}\dots[\vec{\mathbf{B}}]\vec{v_1}(p_0)) - ([\vec{\mathbf{B}}]\vec{v_{n-1}}\dots[\vec{\mathbf{B}}]\vec{v_1}(p_0)) = 0$$

$\Rightarrow \exists k < \infty$ s.t. $[\vec{\mathbf{B}}]\vec{v_n}\dots[\vec{\mathbf{B}}]\vec{v_1}(p_0) \approx [\vec{\mathbf{B}}]\vec{v_{n-1}}\dots[\vec{\mathbf{B}}]\vec{v_1} \Rightarrow [\vec{\mathbf{B}}]\vec{v_n} \approx \mathbf{I}$.

(B) The sequence converges to a certainty (i.e. either $p = 0$ or $p = 1$) that reflects the true underlying difference in the distribution of military weapons between the two countries.

$$\lim_{n\to\infty} p_n = s, s \in 0, 1$$

These two properties can be observed using the following R code, where we simulate different p_n sequences that draw from a range of underlying differences between the distribution of weapons possessed by each side.

R CODE: MULTIPLE SIMULATIONS OF BAYESIAN UPDATING

These plots illustrate a key takeaway that, with underlying distributions that closely resemble each other (i.e. the U.S. and North Korea are similarly matched), it requires a higher number of n interactions for the sequence of posteriors to converge to certainty. In real life, since such interactions and confrontations are risky and costly, it makes sense to minimize n. As much information as possible should be extracted from the resulting p_n, not only in terms of whether p_n is closer to 0 or to 1 (which informs which country is stronger), but also how far it is from 0 or 1 (the farther away it is, the more likely that the sequence would take lots of interactions to converge, suggesting that the two countries are more evenly matched).

Appendix

Simulating the War Card Game

We are not the first to undertake this exercise, and we started from the R source available from PremierSoccerStats blog. But we soon ran into a problem: The code as originally written generates games that result in a stalemate. The game goes on forever, after reaching a point where both players have 26 cards, and from that point on each side wins one play in an infinite, alternating sequence.

There are two alternative solutions to this:

1. Interrupt the loop and call the game a tie.
2. Introduce randomness in every play.

Solution 1

The original author settled for solution #1 and interrupted the loop after an arbitrary, fixed number of plays (5000). For our purposes this solution is unsatisfactory, as the repetition involved in ties could skew the frequency distribution of our Markov process. In order to recognize the stalemate situation earlier, we derived a solution from the definition of the limit of a sequence s_n.

$$\forall \epsilon > 0,\ \exists N \text{ such that } \forall n > N\ |s_n - s| < \epsilon$$

Given:

- r is the range of the number of cards in a hand over the last 52 plays
- n is the running total number of plays

Define the sequence s_n as:

$$s_n = \frac{n^{|2-r|}}{n^{\frac{2}{r}}}$$

Then within 52 plays after reaching a stalemate condition, the sequence converges, approaching a limit of 0.

While we drew inspiration from this lesson in real analysis, the resulting R code improves readability by maintaining the discrete values.

```
limit = length(deck) / 2
# avoid infinite loop with an approach deriving from the "limit of a sequence" definition
# > For all epsilon, there exists an N such that for all n > N, |s_n - s| < epsilon
# In this case, if in the last 52 plays the number of cards are within 1
# of the limit of 26 (half the deck), the game has converged to the limit (a tie)
# and will last forever if we let it.
if (length(p1Cards) > length(deck) && all(tail(p1Cards,length(deck)) %in% ((limit-1):(limit+1)) )) {
  print(paste("Infinite game:", game$id, ", plays:", game$plays))
  game$infinite=TRUE
  break
}
# the original circuit breaker, retained here to verify our improvement
else if (length(p1Cards) > 5000) {
  print(paste("Infinite game not caught by limit, game:", game$id, ", plays:", game$plays))
  game$infinite=TRUE
  break
}
```

Solution 2

Infinite games are, to the relief of many a War player, very unlikely in practice. They strike only in simulations, because the infinite condition arises only when cards maintain their ordering after the original deal.

To understand how this situation arises, first consider that the only aspect of the War card game where a player influences the outcome is in the replacement of cards to the bottom of her deck after a winning play. If the cards are replaced in the same, deterministic order every time (as is unlikely to happen with humans, but likely to happen in a naive computer simulation), then certain original hand configurations can result in an infinite cycle.

A solution that closely resembles what happens in real life is to randomize the order of cards when replacing them to the bottom of the deck. We made that improvement for the simulated data used in this report, using the R function `sample`.

```
# Code for replicating War card Game
# from https://www.r-bloggers.com/simulated-war/
# and http://www.premiersoccerstats.com/wordpress/?p=825&utm_source=rss&utm_medium=rss&utm_campaign=sim

# Andrew Clark April 01 2012
# Modifications by Emmet Murphy December 2017

library(sets)

# make simulation replicable
set.seed(1068)

games <- data.frame(id=numeric(), strength=numeric(),strength_pct=numeric(), aces=numeric(),faces=numeri
deckSizes <- data.frame(id=numeric(),play=numeric(),p1Cards=numeric())
i <- 1
game_count <- 100
for (i in 1:game_count) {
    # create a regular deck.
    # All suits are equivalent so there will be four of each number
    deck <- rep(2:14,4)
    deck_strength <- sum(deck)

    assign("p1", sample(deck,26, replace=FALSE))

    diffs <- gset_difference(as.gset(deck), as.gset(p1))
    # create vector
    p2 <- rep(unlist(diffs), times=gset_memberships(diffs))
    # this produces the right cards but in order so randomize
    assign("p2", sample(p2,26, replace=FALSE))
    #p1
    # [1]  3  3  6 14 10 13  3 10 12  5 11  8  2  5  8  4 10  5  8  4
    # [21]  8  7  7  7  9 14
    #p2
    # [1] 12  6  9  5  9  9 13  4  2 14  4 13 13  6  6 11  7 11 12  2
    # [21] 10  3  2 11 12 14
    p1Cards <- length(p1)
    strength <- sum(p1) # 196 total of players is always 416 so p2 has stronger hand
    strength_pct <- strength / deck_strength
    aces <- sum(p1==14) # 2
    faces <- sum(p1 >= 10 && p1 <=13)
    deuces <- sum(p1==2) # 1
    result <- "N"

    game <- data.frame(id=i, strength=strength,strength_pct=strength_pct,aces=aces,faces=faces,deuces=d

    draw <- c(p1[1],p2[1])

    booty <- c()

    repeat { # for each match of cards
        game$plays <- game$plays + 1
        if (p1[1]>p2[1]) {
            if (p1[1]==14&p2[1]==2){ #  ace(14) vs a 2
```

```
            p2 <- c(p2[-1],sample(draw))
            p1 <- p1[-1]
        } else {
            p1 <- c(p1[-1],sample(draw))
            p2 <- p2[-1]
        }
    } else if (p2[1]>p1[1]) {
        if (p2[1]==14&p1[1]==2){
            p1 <- c(p1[-1],sample(draw))
            p2 <- p2[-1]
        } else {
            p2 <- c(p2[-1],sample(draw))
            p1 <- p1[-1]
        }
    } else {
        while (p1[1]==p2[1]) {

            # need at least 5 cards to play game
            if (length(p1)<5|length(p2)<5) {
                break
            }
            # displayed card plus next three from each player
            booty <- c(booty,p1[1],p1[2],p1[3],p1[4],p2[1],p2[2],p2[3],p2[4])

            #  remove these cards from the p1,p2 so that new p1[1] is next shown
            p1 <- p1[-(1:4)]
            p2 <- p2[-(1:4)]
        }

        draw <- c(p1[1],p2[1])

        if (p1[1]>p2[1]) {
            p1 <- c(p1[-1],sample(booty),sample(draw))
            p2 <- p2[-1]
        } else {
            p2 <- c(p2[-1],sample(booty),sample(draw))
            p1 <- p1[-1]
        }
    }
    #  battle over

    # keep running total of deck size
    p1Cards <- c(p1Cards,length(p1))

    # test for game over
    if(length(p1)==52 || length(p1)==0){
        break
    }

    # reset for next iteration
    booty <- c()
    draw <- c(p1[1],p2[1])
}
```

```
    # war over

    if (length(p1) == 0) {
        game$result = "L"
    } else if (length(p1) == 52) {
        game$result = "W"
    } else if (game$infinite) {
        game$result = "T"
    } else {
        stop(paste("Unclear result with p1 cards: ", length(p1)))
    }

    games <- rbind(games,game)
    deckSize <- data.frame(id=i,play=1:length(p1Cards),p1Cards=p1Cards)
    deckSizes <- rbind(deckSizes,deckSize)
}

# save for later analysis
write.csv(games,paste("games", game_count, "random.csv", sep=''))
write.csv(deckSizes,paste("deckSizes", game_count, "random.csv", sep=''))
```

Summary of Requirements Met

Basic

- Topic is related to linear algebra and real analysis
- The script is commented well enough to be reasonably self-explanatory.
- The script generates at least one nice-looking graphic.
- The script is uploaded before noon, and both team members participate in the presentation session.

Additional

- Project is related to a topic that you have studied in another course this term.
- Uses an R function that has not appeared in any Math 23 lecture script.
- Uses a random-number function and a for loop.
- Incorporates ideas both from linear algebra and from real analysis.
- Defines and uses at least two functions.
- Includes two related but distinct topics.
- Professional looking software engineering
- Integrating well-written LaTeX with R code in a final write-up via R markdown