A Study on Card Shuffling via Simulations

The arts and science of card shuffling

The arts and science of card shuffling have been in the interest of statisticians and gaming enthusiasts for centuries. Card games, for leisure or competitions, depend heavily on the integrity and randomness from the shuffling process. The efficiency of shuffles directly influences the fairness of games, making it a focus of great importance in various domains, from casual games to high-stakes tournaments.

This study embarks on a comprehensive exploration of various card shuffling techniques, aiming to unravel the extent of randomness each method introduces. Our investigation covers three traditional techniques, namely, riffle shuffle, overhand shuffle, and pile shuffle. The primary objective is to quantify and compare the degree of randomisation achieved by each technique, shedding light on their respective strengths and weaknesses.

Understanding the dynamics of card shuffling is not merely an academic pursuit; it holds practical implications in ensuring the fairness and unpredictability of card games. The study employs a multifaceted approach, integrating simulations, mathematical models, statistical analyses, to evaluate the randomness achieved by various shuffling methods.

Riffle shuffle

Introduction and set-up

The riffle shuffle is a classic way of mixing playing cards. Roughly speaking, riffle shuffle involves splitting a deck of cards into two (not necessarily equal) halves, then interleaving the cards together, making a “waterfall effect” as they mix. In this study, we will base our simulations primarily on the techniques and theory introduced the article “How Many Times Do I Have To Shuffle This Deck?” (http://www.ams.org/publicoutreach/feature-column/fcarc-shuffle) (we will address it as “the AMS article” for convenience in the rest of the study), adding some results from related works of literature where applicable.

Simulation: The actual riffle shuffle

In the AMS article, the author, Austin, introduced two ways of simulating a round of riffle shuffle that effectively give the same result. We start by exploring “the actual” riffle shuffle via simulations, following the terminology and procedures given in the article with more comments and details.

For the following simulations, we first create a deck function to generate a vector consisting of the numbers 1-n (in order) to represent the initial deck of cards. Think of this action as using a marker pen to decorate the n cards with indexes from 1 to n respectively based on their original position for identifiability. In practice, we will have “Card 1, Card 2, …, Card n” (the output of the function `deck`) in “Position 1, Position 2, …, Position n” (the natural order).

```
deck <- function(n){seq(1,n,1)}
#this returns a series of numbers from 1 to n
```

Below is a function where we determine how many and what cards we are choosing. Following the convention here (http://www.ams.org/publicoutreach/feature-column/fcarc-shuffle), we may think of it as putting the chosen cards in our left hand, hence naming the function `lefthand_r` , where r denotes “riffle”. The input will be the total number of cards (n). Since we are choosing the top c cards, Card 1, Card 2, …, Card c will be chosen. The output of our next function is the indexes of the cards we have chosen. Note that c can be zero such that the probabilities add to 1.

Note that naturally, Card c+1, Card c+2, …, Card n will be on our right hand.

```
lefthand_r <- function(n){
  c <- sample((0:n), size = 1, prob = choose(n, 0:n) / (2^n))
  #this is the total number of cards we will have on our left hand, where the probability that we choose c cards
is given by the function "choose(n, c)/(2^n)", as we are choosing according to the binomial distribution, based o
n the AMS article
  seq(1,c,1)
  #we are choosing c cards "from the top of the deck". Hence, one we know that c cards will be chosen, we automat
ically know that Card 1, Card 2, ..., Card c are the c cards to be placed on our left hand.
}
```

After we have put c cards (Card 1, Card 2, …, Card n) into our left hand, our next action is to placed the c cards into random positions. The next function helps us decide the set of c positions out of the n cards.

```
pos_r <- function(n, c){
  sort(sample(seq(1,n,1),c))
  #we select c elements from the sequence 1 to n and sort them in ascending order
}
```

We then place the chosen cards (from the left hand) into the chosen positions in order, and the remaining (n-c) cards (from the right hand), in order, in the other positions.

With the previous simpler functions, we can now create a single riffle shuffle function `rs` with input “deck” (a set of marked deck of cards) and output f, which is a vector of length n denoting the new deck of cards sitting on Position 1, Position 2, …, Position n in order.

For example, if we start with 6 cards (assuming we have the deck “1 2 3 4 5 6”) and randomly choose 2 cards to be placed in the left hand (assuming c=2, `lefthand_r` would return “1 2”), one of the possible random positions we would have could be Position 2 and Position 4 (assuming that `pos_r` returns “2 4” in a particular case). This means that we place the Card 1 into Position 2, Card 2 into Position 4; for the remaining card, we have Card 3 - Position 1, Card 4 - Position 3, Card 5 - Position 5 and Card 6 - Position 6. We would want our function `rs` to return “3 1 4 2 5 6”, representing “Card 3, Card 1, Card 4, Card 2, Card 5, Card 6” in “Position 1, Position 2, Position 3, Position 4, Position 5, Position 6”.

```
rs <- function(deck){
  n <- length(deck)
  f <- rep(-1,n) #we create n positions with placeholders -1s and will update this later


  lefthand <- deck[lefthand_r(n)] #we choose the c cards to be placed on our left hand
  righthand <- setdiff(deck, lefthand) #this is, naturally, the remaining cards on the right hand

  c <- length(lefthand) #for convenience


  pos <- pos_r(n, c) #these are the new positions for the chosen c cards in the left hand
  remaining <- setdiff(deck,pos) #these are the remaining positions for the n-c cards in the right hand

  #we now fill f with cards from our left hand first
  for (i in (1:c)){
    f[pos[i]] <- lefthand[i]
  }

  #and fill f with cards from our right hand
  for (i in (1:(n-c))){
    f[remaining[i]] <- righthand[i]
  }

  f
}
```

Then, we create the following function `mult_r` where we perform multiple riffle shuffles on a deck of cards. The input is the deck we start with (“deck”) and the number of shuffles we want (“N”). The output will be the final positions of the deck after n shuffles.

```
mult_r <- function(deck,N){ #perform riffle shuffle N times to a deck of cards
  for (i in (1:N)){
    deck <- rs(deck)
  }
  deck
}
```

We will return to assessing the actual riffle shuffle method in the section “The magic number seven” along with the inverse riffle shuffle method.

Simulation: inverse riffle shuffle

We now move on to the second method introduced in the AMS article. Austin thinks it is an easier way to help us inspect what happens in a riffle shuffle.

The first function we create will be to resemble the act of decorating every card in a deck with a “0” or “1”, which is simply to draw n values from a Binomial (1,0.5) distribution, or, Bernoulli.

```
label01_r <- function(n){
  rbinom(n,1,0.5)
}
```

Then, we use the label to divide the cards into two ordered groups with the group labeled “0” on top.

```
irs <- function(deck){
  f <- c() #we will update this collection later

  n <- length(deck)

  label <- label01_r(n) #we label some of the n cards with 1
  zeros <- n-sum(label) #naturally, the rest of the cards will be labeled with 0

  for (i in (1:n)){
    if(label[i]==0){f<-c(f,deck[i])} #we fill "f" with the cards labelled 0 first
  }

  for (i in (1:n)){
    if(label[i]==1){f<-c(f,deck[i])} #then we fill "f" with the cards labelled 1 last
  }

  f

}
```

For multiple inverse riffle shuffles, we have the following function. Similarly, the deck we start with (“deck”) and the number of inverse riffle shuffles we want (“N”). The output will be the final positions of the deck after n inverse riffle shuffles.

```
mult_ir <- function(deck,N){ #doing inverse riffle shuffles N times to a deck
  for (i in (1:N)){
    deck <- irs(deck)
  }
  deck
}
```

In the following section, we will briefly verify and address the key points from the study of riffle shuffle mainly through simulations.

The magic number seven - or twelve?

We now consider the idea of **distance between two decks of cards** as it is a convenient way for us to quantify the extent of randomisation through simulations. We would like to strongly underscore that this is a different concept from the notion of *distance between probability densities*, introduced in the AMS article, which is mainly useful for theoretical investigation.

In particular, for our study, we define the distance between two decks of cards (of the same length) as the sum of the Euclidean distances between the labels of two cards in the same position. Notationwise, if we have deck X "$x_1 x_2 \dots x_n$" represented by a labeled sequence "$x_1, x_2, \dots, x_n$" and deck Y "$y_1 y_2 \dots y_n$" represented by a labeled sequence "$y_1, y_2, \dots, y_n$" according to our convention in the set-up, then we denote the distance between the two decks of cards as

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}\,.$$

For example, the distance between the deck "1 2 3" (represented by the sequence "1,2,3") and the deck "3 2 1" (represented by the sequence "3,2,1") is

$$\sqrt{(1-3)^2 + (2-2)^2 + (3-1)^2} = 2\sqrt{2}.$$

As Euclidean distance is a widely used distance metric, this is a good way to represent "how different" two deck of cards are. Provided we have an initial deck of cards, the more "different" the final deck card is from the initial deck of cards, the more randomised the shuffles are. Clearly, our aim is to maximise the distance between two decks of cards with as little number of shuffles as possible.

Following this definition, we provide a function that helps us find the distance between two particular decks below.

```
distance_r <- function(deck1, deck2){
  n <- length(deck1) #note that deck1 and deck2 should have the same lengths
  distance <- 0 #this will be updated as we gradually add the distances between each position.
  for (i in (1:n)){
    distance <- distance + sqrt((deck1[i]-deck2[i])^2)
  }
  distance
}
```

Without loss of generality, we investigate for a deck of 52 cards following the natural order, i.e. the deck "1 2 3 … 51 52".

```
whole <- deck(52)
```

The following function then helps us find the mean distance between the final deck and the initial deck after we perform N riffle shuffles on the same initial deck. We replicate for $t = 1000$ for better accuracy.

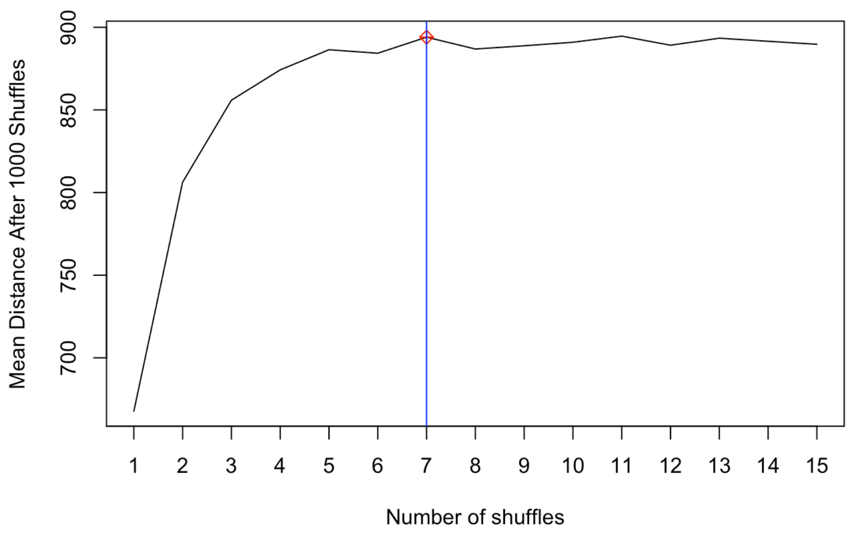
```
mean_distance_r <- function(deck,N,t=1000){
  #deck is the initial ordering
  #N is the number of shuffles we perform on the initial deck;
  #t is the number of times we want to replicate the simulations, set it to 100 by default
  n <- length(deck)
  M <- matrix(data=rep(-1,t*n),nrow=t) #create a matrix for storing the results for different simulations
  for (i in (1:t)){
    final_deck <- mult_r(deck,N)
    M[i,] <- final_deck
  } #when this loop is finished, we will have a matrix M with the results of all the final decks from t replicati
ons

  #now we calculate the average difference
  total_distance <- 0
  for (i in (1:t)){
    total_distance <- total_distance+distance_r(deck,M[i,])
  }
  total_distance/t
}
```

We now find the mean distances for $N = 1, 2, \dots, 15$ times for the initial deck "whole" (1 2 … 51 52)

```
set.seed(160224)
md1rs <- mean_distance_r(whole,1)
md2rs <- mean_distance_r(whole,2)
md3rs <- mean_distance_r(whole,3)
md4rs <- mean_distance_r(whole,4)
md5rs <- mean_distance_r(whole,5)
md6rs <- mean_distance_r(whole,6)
md7rs <- mean_distance_r(whole,7)
md8rs <- mean_distance_r(whole,8)
md9rs <- mean_distance_r(whole,9)
md10rs <- mean_distance_r(whole,10)
md11rs <- mean_distance_r(whole,11)
md12rs <- mean_distance_r(whole,12)
md13rs <- mean_distance_r(whole,13)
md14rs <- mean_distance_r(whole,14)
md15rs <- mean_distance_r(whole,15)
```

We then plot the graph of mean distances against the number of shuffles.

```
plot(y=c(md1rs,
         md2rs,
         md3rs,
         md4rs,
         md5rs,
         md6rs,
         md7rs,
         md8rs,
         md9rs,
         md10rs,
         md11rs,
         md12rs,
         md13rs,
         md14rs,
         md15rs), x = seq(1,15,1), type="l",
     xaxt = "n",
     ylab = "Mean Distance After 1000 Shuffles",
     xlab = "Number of shuffles")

axis(side = 1, at = seq(0,15,1))
points(x=7,y=md7rs, pch=9, col = "red")
abline(v=7, col="blue")
```



From the graph, we can see that $N = 7$ is a reasonable number of shuffles we should perform to maximise the mean distance without shuffling too many times.

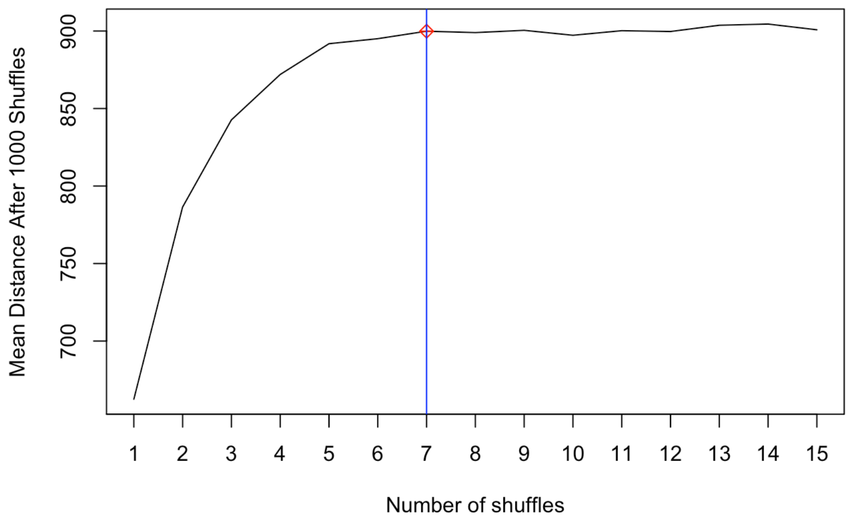
We now do the same thing to inverse riffle shuffle, which is in theory (according to the AMS article), equivalent to the actual riffle shuffle.

Calculate the distance for inverse riffle shuffle:

```
mean_distance_ir <- function(deck,N,t=1000){ #effectively the same coding technique that we have used for riffle
shuffle
  n <- length(deck)
  M <- matrix(data=rep(-1,t*n),nrow=t)
  for (i in (1:t)){
    final_deck <- mult_ir(deck,N)
    M[i,] <- final_deck
  }
  total_distance <- 0
  for (i in (1:t)){
    total_distance <- total_distance+distance_r(deck,M[i,])
  }
  total_distance/t
}
```

```
set.seed(1602241)
md1irs <- mean_distance_ir(whole,1)
md2irs <- mean_distance_ir(whole,2)
md3irs <- mean_distance_ir(whole,3)
md4irs <- mean_distance_ir(whole,4)
md5irs <- mean_distance_ir(whole,5)
md6irs <- mean_distance_ir(whole,6)
md7irs <- mean_distance_ir(whole,7)
md8irs <- mean_distance_ir(whole,8)
md9irs <- mean_distance_ir(whole,9)
md10irs <- mean_distance_ir(whole,10)
md11irs <- mean_distance_ir(whole,11)
md12irs <- mean_distance_ir(whole,12)
md13irs <- mean_distance_ir(whole,13)
md14irs <- mean_distance_ir(whole,14)
md15irs <- mean_distance_ir(whole,15)
```

```
plot(y=c(md1irs,
         md2irs,
         md3irs,
         md4irs,
         md5irs,
         md6irs,
         md7irs,
         md8irs,
         md9irs,
         md10irs,
         md11irs,
         md12irs,
         md13irs,
         md14irs,
         md15irs), x = seq(1,15,1), type="l",
     xaxt = "n",
     ylab = "Mean Distance After 1000 Shuffles",
     xlab = "Number of shuffles")

axis(side = 1, at = seq(0,15,1))

points(x=7,y=md7irs, pch=9, col = "red")
abline(v=7, col="blue")
```



Similarly, $N = 7$ seems like a good number of shuffles we shall perform to maximise randomisation in terms of the distance between decks of cards by our definition.

A little dive into the theory & the controversy

While it is a straightforward way to use the mean distance after a large number of shuffles from simulations to reflect the optimal number of shuffles to increase randomisation, we also acknowledge and replicate some of the theoretical results widely known in the field of riffle shuffle studies.

Replicating the works in the AMS article with added explanations

For an integer c from 1,2,…,n, the probability of a simple value chosen is

$$\mathbb{P}(C = c) = \frac{1}{2^n}\binom{n}{c}.$$

If the first card is selected to put in the left hand with probability c/n, the second card is selected to put in the left hand will have probability $(c-1)/(n-1)$. Similarly, if the first card is selected to put in the right hand with probability $(n-c)/n$, the second card is selected to put in the right hand will have probability $(n-c-1)/(n-1)$.

Austin brings forward the notion of distance between two probability densities Q_1 and Q_2 as

$$||Q_1 - Q_2|| = \max_{A \subset S_n} |Q_1(A) - Q_2(A)|,$$

where

$$|Q_1(A) - Q_2(A)| = \frac{1}{2} \sum_{\pi \subset A} |Q_1(\pi) - Q_2(\pi)|.$$

A is a subset of S_n, “the set of all orderings of the deck”.

Austin claims that we need to estimate how far away from the uniform distribution we are after k shuffles to assess the randomisation, so we strive to understand the distance $Q^k - U$, where $U(\pi) := 1/n!$ is the probability in which each ordering π is equally likely. We would like to comment that essentially, $U(\pi)$ is the probability that we obtain an ideal final deck (maximal randomisation under Austin’s context). Therefore, we would actually want to minimise the distance between probability densities as we want to be as close to the ideal state as possible!

By using results on stopping times and conditional probabilities, Austin has found an upper bound for the distance in question. In particular, he finds that

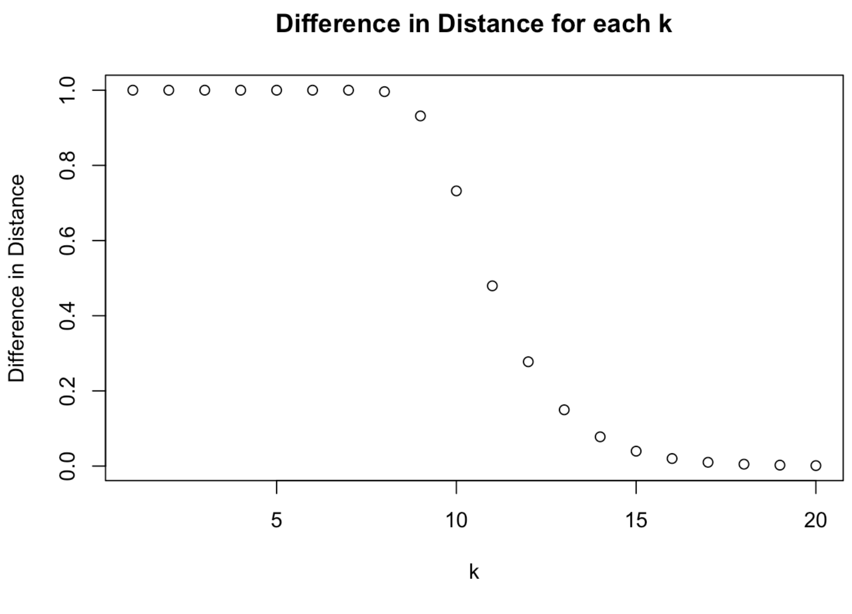
$$||Q^k - U|| \leqslant \mathbb{P}(T > k) = 1 - \prod_{i=1}^{n-1}(1 - \frac{i}{2^k}).$$

We consider shuffling a typical deck of n=52 cards in order using riffle shuffles. The following plot displays the bounds on the difference in distance between probability densities (again, this is not the notion of distance we were using previously for decks of cards) for different times of shuffling according to Austin’s finding. We want to find the respective distances between probability densities for 1,2,…,k riffle shuffles for a deck of n cards.

```r
diffdist_rs <- function(k,n){
  k_list <- c() #we will update this later
for (j in (1:k)) {
  k <- j
  new_list <- c()
  for (i in (1:(n-1))){
    new_term <- 1- i/(2^k) #this is the individual term we have in the pi multiplication
    new_list <- c(new_list,new_term)
  }
  new_k <- 1 - prod(new_list)
  k_list <- c(k_list, new_k)
}
  k_list
}
```

```r
kl <- diffdist_rs(20,52)
```

```r
plot(seq(20),kl, main="Difference in Distance for each k", xlab="k", ylab="Difference in Distance")
```



Austin asserts that under his method, $k = 12$ is a good recommendation for the number of shuffles based on the graph. However, he also finds out later that there actually exists an improved upper bound provided by a more sophisticated study by Diaconis and Bayer (https://www.jstor.org/stable/2959752) (we will call it as “the D & B article”) which gives $k = 7$ as the upper bound. In fact, seven is actually the more accepted answer for the study of riffle shuffles. Clearly, the number seven is more consistent with our simulation results!

Replicating the works in the D & B article

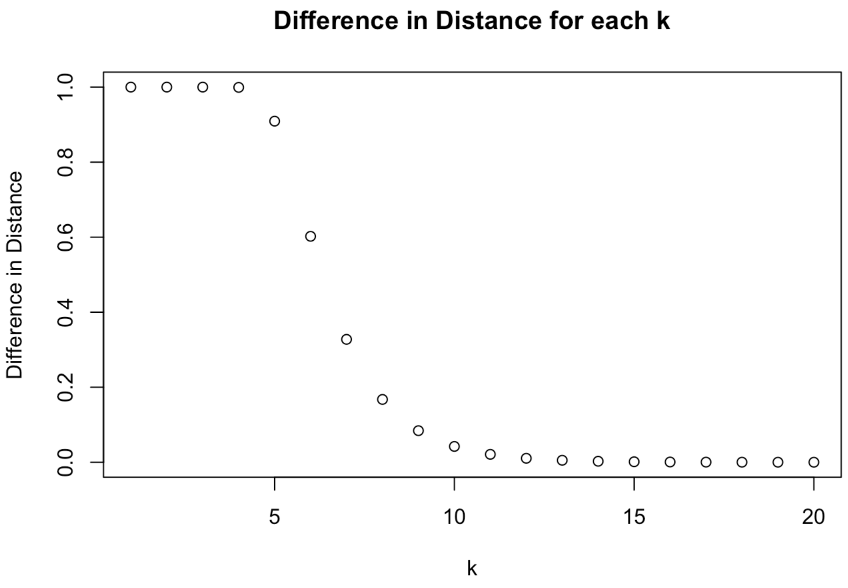
Inspired by Bayer and Diaconis (1992) (https://www.jstor.org/stable/2959752), we have an improved upper bound

$$||Q^k - U||^* = 1 - 2\Phi(\frac{-2^{-\theta}}{4\sqrt{3}}) + O(\frac{1}{n^{1/4}}).$$

```
diffdist_rs1 <- function(k,n){
  k_list <- c()

for (j in (1:k)) {
  k <- j
  theta <- j-1.5*log2(n)
  k_list <- c(k_list, 1-2*pnorm((-2^(-theta))/(4*sqrt(3)))) #ignoring the negligible big-O term
}
  k_list
}
```

```
kl1 <- diffdist_rs1(20,52)
```

```
plot(seq(20),kl1, main="Difference in Distance for each k", xlab="k", ylab="Difference in Distance")
```



From the results given in the work by Diaconis and Bayer, we see that seven is indeed a more reasonable (and efficient!) number of shuffles we have for riffle shuffle.

Overhand shuffle

Introduction and set-up

We consider a deck and separate it into k packets, resulting in k-1 cut points. Each packet of the deck contains random number of cards. The overhand shuffle is such that holding the deck in your right hand and transferring packets from the top of the deck until all cards are transferred to your left hand. Consequently, the positions of the packets reverse, while the positions of the cards within each packets remain unchanged

For example, let’s examine a deck with 7 cards, numbered 1,2,3,4,5,6,7 from the top of the deck. If we would like to separate the deck into 3 packets, which means we have 2 cut points, assuming we have card 1 and 2 in the first packet, card 3 and 4 in the second packet, all other cards in the third packet, that is $12\|34\|567$. After performing one overhand shuffling, the cards in position $567\|34\|12$. As observed, the overhand shuffle reserves the positions of packets but the positions of card within each packet do not change.

A paper by Pemantle (1989) (https://www.math.upenn.edu/~pemantle/papers/overhand2.pdf) indicates that at least 50 overhand shuffles are required to adequately mix the deck, but computer experiments (Pemantle, 1989, p. 49) indicate that in many situations 1,000 or more overhand shuffles are required to randomize a deck. Here, we will based our theories on Rick Wicklin (https://blogs.sas.com/content/iml/2018/09/19/overhand-shuffle-riffle.html) simulations and findings.

Simulation: Overhand shuffle

We first create a deck function to generate the initial deck of n cards. We order and mark the positions of these cards by number 1 to n. We generated a example deck with 52 cards for easier simulation later.

```
deck_function <- function(n){seq(1,n,1)}
deck <- deck_function(52)
```

We now define a function to reverse the position of packets:

```
Overhand <- function(deck, p = 0.5) {
  N <- length(deck) #find total number of cards in the deck
  b <- rbinom(N-1,1,0.5) #simulate cut points
  cutpt <- c(0, which(b == 1), N)  # Cut points in range 1:(N-1)

  if (all(b == 0)) return(deck)    # No cut points were chosen

  rev <- N - cutpt + 1             # Reversed positions

  newDeck <- integer(N)
  for (i in 1:(length(cutpt) - 1)) {
    R <- seq((1 + cutpt[i]), cutpt[i + 1])  # Positions of cards in the right hand
    L <- seq(rev[i + 1], rev[i] - 1)       # New positions in the left hand
    newDeck[L] <- deck[R]                  # Transfer packet from right to left hand
  }

  return(newDeck)  # Return same shape as input
}
```

Recalled that we need to separate the deck into k packets with k-1 cut points. In the simulation, suppose we have total number of N cards in the deck, then we have N-1 positions between adjacent cards and all these N-1 positions could be a cut point or not. The idea distribution to model each position between adjacent cards is Bernoulli distribution with probability =0.5, stated by Rick Wilklin as the ideal typical size of the packets.

If the total number of cards N is large, the probability of simulating no cut point from Bernoulli distribution with p=0.5 is very small. However, when N is very small, for example N=5, event of simulating no cut point is very likely to happen, but in this case, it is meaningless of using overhand shuffle because your total number of cards N is too small, so we return the original deck in this case. If cut points were chosen, we then define a for loop to reverse the positions of packets and finally return the new deck after one shuffling.

Define a function to perform n times overhand shuffle:

```
mult_Overhand <- function(deck, n){
  for(i in (1:n)){
    deck <- Overhand(deck)
  }
  deck
}

# Example usage:
n <- 11
multioverhand<-mult_Overhand(deck, n)

cat("Original Deck:", deck)
```

```
## Original Deck: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
```

```
cat("\nDeck after",n ,"overhand shuffling:", multioverhand)
```

```
##
## Deck after 11 overhand shuffling: 44 46 43 35 47 51 50 45 52 48 41 40 36 39 37 32 21 49 38 42 33 34 17 30 31 2
8 12 27 22 18 29 20 2 23 24 16 11 25 26 8 6 19 14 4 1 7 10 3 9 13 5 15
```

A different magic number?

We would like to reuse the distance between decks function we have used previously "distance_r" in calculating the following.

```
distance_r(deck, mult_Overhand(deck,11))
```

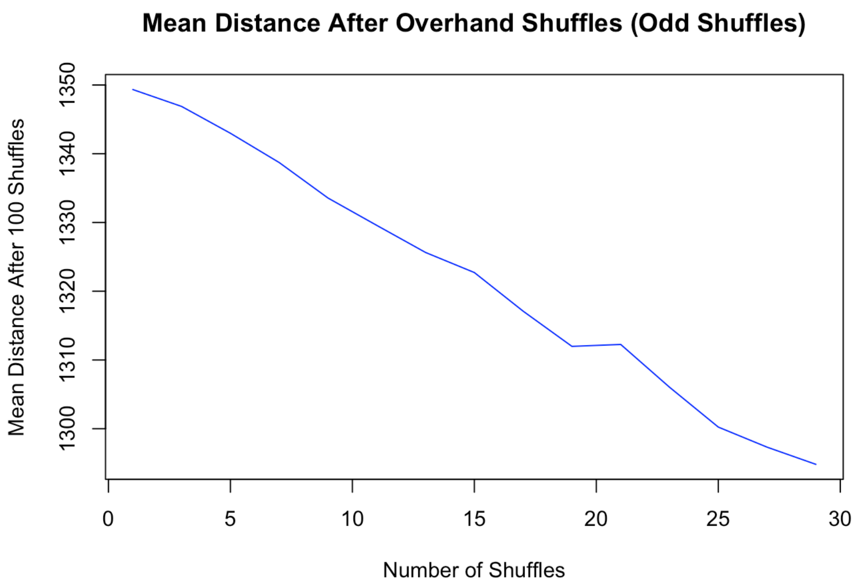
```
## [1] 1346
```

```
distance_r(deck, mult_Overhand(deck,12))
```

```
## [1] 278
```

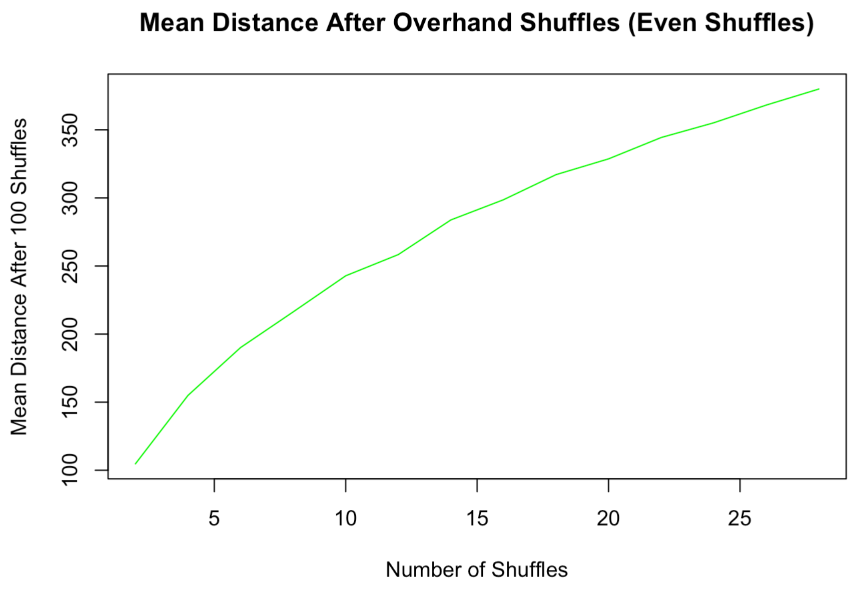
We have calculated the distances for the example deck after 11 and 12 overhand shuffles. The significant difference between distance values is due to the nature of the overhand shuffle, that is you reverse the positions of packets and therefore cards on the top of the deck would be transferred to the bottom of the deck. Consequently, if you perform an odd number of overhand shuffles, the distance would be larger comparing to that of performing an even number.

```
mean_distance_over <- function(deck, N, t = 1000) {
  n <- length(deck)
  M <- matrix(data = rep(-1, t * n), nrow = t)

  for (i in 1:t) {
    final_deck <- mult_Overhand(deck, N)
    M[i,] <- final_deck
  }

  total_distance <- 0
  for (i in 1:t) {
    total_distance <- total_distance + distance_r(deck, M[i,])
  }

  total_distance / t
}

mean_distance_odd <- function(deck, numShuffles, numSimulations = 100) {
  odd_shuffles <- seq(1, numShuffles, by = 2)
  mean_distances_odd <- numeric(length(odd_shuffles))

  for (i in seq_along(odd_shuffles)) {
    mean_distances_odd[i] <- mean_distance_over(deck, odd_shuffles[i], numSimulations)
  }

  return(list(Shuffles = odd_shuffles, MeanDistances = mean_distances_odd))
}

# Example usage:
set.seed(160224)
result_odd <- mean_distance_odd(whole, 29, 100)

# Plot the mean distances for odd shuffles
plot(x = result_odd$Shuffles, y = result_odd$MeanDistances, type = "l", col = "blue", xlab = "Number of Shuffle
s", ylab = "Mean Distance After 100 Shuffles", main = "Mean Distance After Overhand Shuffles (Odd Shuffles)")
```



Odd Shuffles: In the context of the overhand shuffle, the largest mean distance can be seen at the first shuffle with mean distance at 1349.34, although, each packets contains consecutive card positions. Odd shuffles exhibit a distinct behavior where the mean distance between card positions decreases with each iteration. This suggests that odd shuffles tend to bring cards back closer to their original positions. While the initial shuffle may disperse the positions, subsequent odd shuffles gradually restore some order to the deck. The sequence of numbers remains recognizable, reflecting a characteristic of the overhand shuffle that balances randomness with a tendency to preserve the original order.

```r
mean_distance_even <- function(deck, numShuffles, numSimulations = 100) {
  even_shuffles <- seq(2, numShuffles, by = 2)
  mean_distances_even <- numeric(length(even_shuffles))

  for (i in seq_along(even_shuffles)) {
    mean_distances_even[i] <- mean_distance_over(deck, even_shuffles[i], numSimulations)
  }

  return(list(Shuffles = even_shuffles, MeanDistances = mean_distances_even))
}

set.seed(123)
result_even <- mean_distance_even(whole, 29, 100)

# Plot the mean distances for even shuffles
plot(x = result_even$Shuffles, y = result_even$MeanDistances, type = "l", col = "green", xlab = "Number of Shuffl
es", ylab = "Mean Distance After 100 Shuffles", main = "Mean Distance After Overhand Shuffles (Even Shuffles)")
```



Even Shuffles: Conversely, the overhand shuffle unfolds differently after an even number of shuffles. Following this pattern, the top of the deck retains a substantial number of cards that were initially at the top, emphasizing a tendency to maintain some semblance of the original order. Even after 24 overhand shuffles, the mean distance increase slowly reaching 355.16,which is very low compared to odd number of shuffles. this means that numerous cards remain in proximity to their initial positions, and adjacent cards remains close together.

Probability Density

Our theoretical framework draws from the methodologies pioneered by Johan Jonasson (https://www.math.chalmers.se/~jonasson/convrates.pdf). According to this framework, the key observation is that substantial packets of cards tend to maintain proximity throughout multiple shuffling iterations. Jonasson's findings establish that for a deck containing N cards, achieving randomness necessitates $\Theta(n^2 log(N))$ overhand shuffles.

Late Wilson's theory on overhand shuffle demonstrated that τ_{mix} is at most of the order $n^2 \log N$, providing the lower bound for the number of shuffles. Here's a simplified explanation (using p=0.5):

Consider a deck $X_{t\,t=0}^{\infty}$, and let ϕ be a right eigenvector of the transition matrix with corresponding real eigenvalue $\lambda = 1 - \gamma$, where $0 < \gamma < \frac{1}{2}$. Consider a pair of cards (i, j). Let Y_i' and Y_j' follow the distributions of Y_i and Y_j independently. We can establish a coupling such that $(Y_i, Y_j) = (Y_i', Y_j')$ under the condition A, indicating that there are at least two marked slots between cards i and j for the $t + 1$ update of the shuffle. If the distance between card i and card j at time t is at least $10 \log n$, then $P(A) = 1 - o(1/n^5)$.

Consequently,

$$E[Y_i Y_j | X_t] = E[Y_{0i} | X_t] E[Y_{0j} | X_t] + o(n^{-5}) = O(n^{-4}).$$

For other pairs (i, j),

$$E[Y_i Y_j | X_t] \le \max_{s \in S} E[Y_i^2 | X_t = s] = \frac{2}{3} \sum_{j=1}^{\infty} \frac{1}{2^j} \sin\left(\frac{2\pi j}{n}\right)^j < 50n^{-2}.$$

Hence,

$$\sum_i \sum_j E[Y_i Y_j | X_t] < n^2 O(n^{-4}) + 20n \log n \cdot 50n^{-2} = O(n^{-2}) + 1000n^{-1} \log n.$$

Wilson's key theorem states that for all $t \le T$, $\| P(X_t \in \cdot) - \pi \|_{\text{TV}} \ge 1 - a$, where T is determined by the properties of ϕ.To establish a lower bound, Wilson employed a coupling technique for pairs of cards (i, j), showing that the expected product $E[Y_i Y_j | X_t]$ is small, and the key theorem with appropriately chosen parameters yielded the desired lower bound: $\tau_{\text{mix}} \ge (1 + o(1)) \frac{1}{16\pi^2} n^2 \log n$

In the general case where slots are marked with probability p, the step size distribution for the random walk described by a single card is $p(1-p)^{|j|}/(2-p)$ for $j \in \mathbb{Z}$. Recalculating with this step size distribution gives:

$$\gamma = (1+o(1))4\pi^2 \frac{1-p^2}{p^2(2-p)} n^{-2}$$

which leads to:

$$\tau_{\text{mix}} \geq \frac{p^2(2-p)}{8\pi^2(1-p^2)} n^2 \log n$$

Using this function, we can now find the lower bound numbers of overhand shuffle as follows:

```
calculate_mixing_time <- function(n, p) {
  tau_mix <- (p^2 * (2 - p)) / (8 * pi^2 * (1 - p^2)) * n^2 * log(n)
  return(tau_mix)
}

# Example usage
n <- 52
p <- 0.5
result <- calculate_mixing_time(n, p)

# Print the corrected result
print(round(result))
```

```
## [1] 68
```

Meaning that using Wilson’s theory, 68 shuffles are needed to randomize overhand shuffle with 52 cards, with p=0.5.

Pile shuffle

Introduction and set-up

Pile shuffling is a card handling technique that involves dealing out a deck of cards into a number of piles and then stacking those piles on top of each other. Unlike shuffling methods designed for randomisation, pile shuffling is more deterministic and serves a different purpose – primarily to visually separate cards and ensure that cards originally adjacent to each other in the deck end up in different piles.

Simulation: pile shuffle

```
deck <- function(n){seq(1,n,1)}
deck(52)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52
```

```
pile_shuffle <- function(deck, num_piles) {
  piles <- split(deck, rep(1:num_piles, length.out = length(deck)))
  shuffled_deck <- unlist(piles, use.names = FALSE)
  return(shuffled_deck)
}
```

The **`pile_shuffle`** function simulates a pile shuffling process. It takes an initial deck and the number of piles to create during shuffling. For each card in the deck, it assigns it to a pile in a cyclic manner, creating subsets based on the number of piles. The piles are then stacked on top of each other, and the function returns the resulting shuffled deck.

```
# Example usage:
initial_deck <- deck(52)
shuffled_deck <- pile_shuffle(initial_deck, 6)
shuffled_deck
```

```
##  [1]  1  7 13 19 25 31 37 43 49  2  8 14 20 26 32 38 44 50  3  9 15 21 27 33 39
## [26] 45 51  4 10 16 22 28 34 40 46 52  5 11 17 23 29 35 41 47  6 12 18 24 30 36
## [51] 42 48
```

Define a function to perform n times pile shuffle:

```r
mult_Pile <- function(deck, num_piles, n){
  for(i in (1:n)){
    deck <- pile_shuffle(deck, num_piles)
  }
  deck
}

# Example usage:
num_piles <- 5
n <- 11
deck <- deck(52)
multipile<-mult_Pile(deck, num_piles, n)

cat("Original Deck:", initial_deck)
```

```
## Original Deck: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
```

```r
cat("\nDeck after",n,"times pile shuffling (",num_piles, "piles ):", multipile)
```

```
##
## Deck after 11 times pile shuffling ( 5 piles ): 1 16 45 10 25 22 5 36 2 18 15 4 43 33 23 46 3 50 37 48 13 20 5
1 8 31 52 11 28 41 34 21 40 44 12 29 19 24 9 14 35 6 30 47 26 27 32 7 39 38 49 17 42
```

Performing multiple pile shuffling iterations with the same number of piles yields a visually separated deck, as cards originally adjacent to each other are redistributed across distinct piles. However, it is crucial to note that pile shuffling is a deterministic process, providing consistent results with repeated iterations using the same initial deck and pile count. While it enhances visual organisation, pile shuffling alone does not achieve true randomisation.

```r
#There is no randomisation during shuffle so the location of card can be predicted
predicted_card <- function(card_index,deck,num_pile) {
  n <- length(deck)
  kk <- seq(1,n, 1) %% num_pile
  for (i in (1:n)) {
    if (kk[i]==0) {
      kk[i] = kk[i] + num_pile
    }
  }
  index <- kk[card_index]
  num <- sum(kk[1:(card_index)]==index)
  # the wanted card is in the (index)th pile of number num
  return(c(index,num))
}
# Example, want to know the location of the 6th card after one shuffle in 5 piles
out <- predicted_card(6,initial_deck,5)
paste("The new location of the original 6th card is the",out[2],"nd in the ",out[1],"st pile")
```

```
## [1] "The new location of the original 6th card is the 2 nd in the  1 st pile"
```

A critical evaluation towards pile shuffling

Pile shuffling, by design, lacks the stochastic nature needed for true randomization. It will not sufficiently mix the cards, especially with a low number of piles. The deterministic nature means that the original order can be preserved to some extent. Each single card can be tracked during shuffling. A player could count so as to get a strong holding when he/she knows the exact input order of the deck and the number of piles. Therefore, it is more of a method for organizing cards to make sure each player get the needed number of cards, rather than introducing randomness. If we want to re-design pile shuffling for introducing randomization, we can choose to randomly locate the card to each pile with probability density $1/N$ when there is N number of piles. However, in practice, these random locating is totally decided by the player who perform shuffling. A human brain may have bias.

General discussion: what the academia thinks of shuffling

The idea of randomisation is highly subjective especially bearing in mind the different purposes people have for shuffling a deck of cards. Like we have stated at the start of the study, it is a combination of arts and science. There are also multiple ways of defining and computing the distances between probability densities. This discrepancy is discussed in further details in the article When is a deck of cards well shuffled (https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjbrdyW3caEAxU2TkEAHQB5DiUQFnoECBgQAQ&url=https%3A%2F%2Frepository.tudelft.nl%2Fislar
b29c-4765-ac15-c35ed0bcd8fc%2Fdatastream%2FOBJ%2Fdownload&usg=AOvVaw1yCly2X8U-ZMZ64--pk93v&opi=89978449)? It is crucial to acknowledge the inherent complexities in the realm of card shuffling. In addition to the discrepancies we have mentioned, we also acknowledge that the skills of the shuffler also contribute to the overall unpredictability of the arts and science of shuffling. This study, while providing valuable insights, is more reflective of an informed perspective rather than an absolute assertion of mathematical facts.

We would like to conclude our investigation via this opinion in this article by Levin, Perez, Wilmer (https://pages.uoregon.edu/dlevin/MARKOV/markovmixing.pdf),

> “Ultimately, the question of how many shuffles suffice for a 52-card deck is one of opinion, not mathematical fact…there exists at least one game playable by human beings for which 7 shuffles clearly do not suffice.”

(Particularly, omitting some onerous details, they claim that in some casino games, 11 or 12 might be a more appropriate number of shuffles for the example of riffle shuffle, rather than the common belief of 7.)