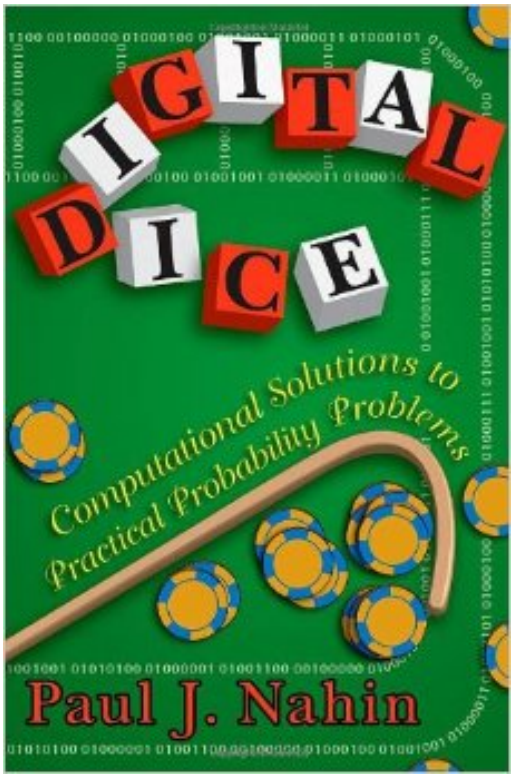
Digital Dice

RK

March 28, 2015

Abstract

This document contains a brief summary and `R` code to solve all the puzzles from the book "Digital Dice".



Contents

Summary

In the last few decades, enormous computational speed has become accessible to many. Modern day desktop has good enough memory and processing speed that enables a data analyst to compute probabilities and perform statistical inference by writing computer programs. In this context, the book can serve as a starting point to anyone who wishes to explore the subject of computational probability.

This book has 21 puzzles that can be solved via simulation. Solving a puzzle has its own advantages. Give a dataset with one dependent variable and a set of predictors to a dozen people asking them to fit a regression model; I bet that you will see at least a dozen models, each of which could be argued as a plausible model. Puzzles are different. There are constraints put around the problem that you are forced to get that ONE RIGHT solution to the problem. In doing so, you develop much more sophisticated thinking skills.

I have thoroughly enjoyed working through this book. In the introductory chapter, the author provides a basic framework for computational probability by showing ways to simulate and compute probabilities. This introductory chapter gives to the reader all the ammunition required to solve the various puzzles of the book. The author provides detailed solutions that includes relevant MATLAB code, to all the 21 puzzles.

Some of my favorite puzzles from the book that are enlightening as well as sometimes paradoxical are :

- The Gamow-Stern Elevator
- The Pipe Smoker's Discovery
- A Toilet Paper Dilemma - I liked this puzzle a lot as it exposed my faulty thinking
- Parrondo's Paradox - This was quite a perplexing result
- How Long Is the Wait to Get the Potato Salad ? The question was easy but my code is clumsy. Need to refine it some day
- The Appeals court Paradox - Another one of those quirky aspects of probability that challenges one's mental models

What's in this document?

I have written `R` code that aims to computationally solve each of the puzzles in the book. For each puzzle, there are two subsections. First subsection spells out my attempt at solving the puzzle. The second subsection contains my learnings from reading through the solution given by the author. The author provides extremely detailed MATLAB code that anyone who has no exposure to MATLAB can also understand. In many cases I found that the code snippets in the book looked like pseudo code, given the elaborate nature of the code.

There are many good references mentioned for each of the puzzle solutions so that interested readers can explore further aspects. In most of the cases, the reader realizes that closed form solutions are extremely tedious to derive and simulation based procedures make it easy to obtain solutions to many intractable problems.

1 The Clumsy Dishwasher Problem

My attempt

The closed form solution for this puzzle is

$$\left\{(1/5)^5 + \binom{5}{1}(1/5)^4(4/5)\right\} = 21/3125 = 0.00672$$

The following single line of code gives the same result via simulation :

```
puzzle_1_results <- mean(replicate(1000000, sum( sample(1:5, 5, replace = TRUE) ==1 ) >=4 ))
print(puzzle_1_results)

## [1] 0.006602
```

If you assume that everyone has equal probability of breaking the dish, the probability of a specific person breaking it is 0.0066. Since this is an extreme small probability, it is highly likely that the identified person is indeed clumsy.

Learning

The author's approach is similar to mine.

2 Will Lil and Bill meet at the Malt Shop?

My attempt

One can solve this via computing the area inside a square which corresponds to the situation that Lil and Bill meet

```
puzzle_2_cf_results_1 <- 1-(23^2 + 25^2)/30^2*0.5
puzzle_2_cf_results_2 <- 1-(25^2 + 25^2)/30^2*0.5
puzzle_2_cf_results_3 <- 1-(23^2 + 23^2)/30^2*0.5
puzzle_2_cf_results   <- c(puzzle_2_cf_results_1, puzzle_2_cf_results_2,
                           puzzle_2_cf_results_3)
```

One can also solve this via simulation

```
meeting_prob          <- function(wt_lil,wt_bill){
  mean(replicate(100000,{
  times               <- runif(2,0,30)
  ifelse(times[1]>times[2], times[1]-times[2] <=wt_bill ,  times[2]-times[1] <=wt_lil )
  }))
}
puzzle_2_sim_results  <- c(meeting_prob(5,7),meeting_prob(5,5),meeting_prob(7,7))
```

```
puzzle_2_results            <- data.frame(puzzle_2_cf_results , puzzle_2_sim_results)
colnames(puzzle_2_results)  <- c("closed form", "simulation")
rownames(puzzle_2_results)  <- c("(5,7)","(5,5)","(7,7)")
```

	closed form	simulation
(5,7)	0.35889	0.35976
(5,5)	0.30556	0.30800
(7,7)	0.41222	0.41185

Learning

The author's approach is similar to mine.

3 A Parallel Parking Question

My attempt

```
mutual_pairs     <- function(n){
  spots          <- sort(runif(n))
  nn             <- c(4,rep(0,(n-2)),2)
  nn[2:(n-1)]    <- ifelse( (spots[2:(n-1)]-spots[1:(n-2)]) >
                              spots[3:n] - (spots[2:(n-1)]), 4, 2)
  sum(nn[1:(n-1)]%/%nn[2:n]==2)*2/n
}
cars             <- c(4:12, 20,30)
prob             <- c(1,2/3,sapply(cars,function(z)mean(replicate(1000,mutual_pairs(z)))))
puzzle_3_results <- data.frame( "probability"= prob)
rownames(puzzle_3_results) <- c(2:12,20,30)
```

	probability
2	1.00000
3	0.66667
4	0.66650
5	0.65800
6	0.67167
7	0.67486
8	0.65550
9	0.66867
10	0.66940
11	0.66145
12	0.66600
20	0.66250
30	0.66433

The simulation results makes one guess that the probability is $2/3$ for every $n > 2$. Indeed it is surprising that whatever be the value of $n > 2$, the probability of finding a mutual nearest neighbor is $2/3$.

Learning

The author is equally surprised with the result and mentions a few references that give closed form solutions to the problem as:

$$\begin{aligned}
\text{probability (1d)} &= 2/3 \\
\text{probability (2d)} &= 6 * \pi/(8\pi + 3\sqrt(3)) = 0.621505 \\
\text{probability (3d)} &= 16/27 = 0.592592
\end{aligned}$$

4 A Curious Coin-Flipping Game

My attempt

```
time_to_ruin <- function(l, m, n , p){
  counter <- c(l, m, n)
  tosses  <- 0
  while(!any(counter==0)){
      trial  <- rbinom(3,1,p)
      tosses <- tosses + 1
      if( sum(trial) == 2 ) {
        idx_1 <- which(trial==1)
        idx_0 <- which(trial==0)
        counter[idx_1] <- counter[idx_1] - 1
        counter[idx_0] <- counter[idx_0] + 2
      } else if (sum(trial)==1){
          idx_1 <- which(trial==1)
          idx_0 <- which(trial==0)
          counter[idx_1] <- counter[idx_1] + 2
          counter[idx_0] <- counter[idx_0] - 1
        }
    }
  tosses
 }
puzzle_4_results <- data.frame( l = rep( c(1,1,2,3,4), times=2),
                                m = rep( c(1,2, 3, 3, 7), times=2),
                                n= rep(c(1,3, 4, 3, 9), times=2),
                                p=c(rep(0.5,5), rep(0.4,5)))

puzzle_4_results$average <- apply(puzzle_4_results, 1, function(z) {
                            mean(replicate(10000,time_to_ruin(z[1], z[2], z[3], z[4])))
                             })
```

l	m	n	p	average
1	1	1	0.50	1.3222
1	2	3	0.50	1.9910
2	3	4	0.50	4.5935
3	3	3	0.50	5.1505
4	7	9	0.50	18.6390
1	1	1	0.40	1.3840
1	2	3	0.40	2.0859
2	3	4	0.40	4.7464
3	3	3	0.40	5.3658
4	7	9	0.40	19.7320

Learning

The author provides a closed form solution for a fair coin as

$$\mu = \frac{4lmn}{3(l+m+n-2)}$$

Hence analytical answers for the puzzle are

l	m	n	p	average	theoretical
1	1	1	0.50	1.3222	1.3333
1	2	3	0.50	1.9910	2.0000
2	3	4	0.50	4.5935	4.5714
3	3	3	0.50	5.1505	5.1429
4	7	9	0.50	18.6390	18.6667
1	1	1	0.40	1.3840	
1	2	3	0.40	2.0859	
2	3	4	0.40	4.7464	
3	3	3	0.40	5.3658	
4	7	9	0.40	19.7320	

The author derives the solution for $l = 1, m = 1, n = 1$ and for any p, the expected tosses for ruin as

$$\mu = \frac{1}{3p(1-p)}$$

Hence out of all possible values for p, for $p = 1/2$, the game comes to end quicky. For $p \to 0$ or $p \to \infty$, it becomes more and more likely that all three coins will show the same face on a toss. Since such an outcome results in no coins changing hands, this raises the number of tosses we expect to see before one of the men is ruined.

5 The Gamow-Stern Elevator Puzzle

My attempt

```
# 0 represent elevator going up, 1 represents elevator going down
waiting_time       <- function(direction, x) {
  if(direction == 0 & x > 1) return(c(1, 11 - x))
  if(direction == 1 & x > 1) return(c(1, x - 1))
  if(direction == 0 & x < 1) return(c(0, 1 - x))
  if(direction == 1 & x < 1) return(c(0, x + 1))
}
n                  <- 100000
realizations       <- data.frame(dx= sample(c(0, 1), n , TRUE),
                         dy= sample(c(0, 1), n , TRUE),
                         x=runif(n, 0, 6), y = runif(n, 0, 6) )
puzzle_5_sim_1     <-  apply(realizations,1, function(z){
                        elevator_1  <- waiting_time(z[1],z[3])
                        elevator_2  <- waiting_time(z[2],z[4])
                        ifelse(elevator_1[2] < elevator_2[2] ,elevator_1[1], elevator_2[1] )
                        })
puzzle_5_results_1  <- data.frame(simulated = mean(puzzle_5_sim_1) , actual = 13/18)
```

simulated	actual
0.7223	0.7222

Three elevator building

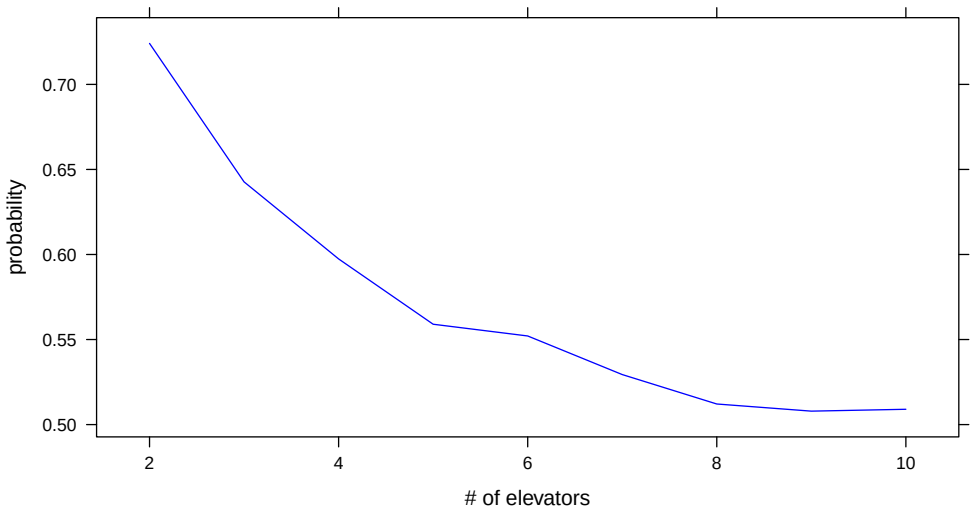
```
n                  <- 100000
realizations       <- data.frame(dx= sample(c(0, 1), n , TRUE),
                         dy= sample(c(0, 1), n , TRUE), dz= sample(c(0, 1), n , TRUE),
                         x=runif(n, 0, 6), y = runif(n, 0, 6), z = runif(n, 0, 6) )
puzzle_5_sim_2     <-  apply(realizations,1, function(z){
                        elevator_1  <- waiting_time(z[1],z[4])
                        elevator_2  <- waiting_time(z[2],z[5])
                        elevator_3  <- waiting_time(z[3],z[6])
                        times       <- c(elevator_1[2], elevator_2[2], elevator_3[2])
                        direction   <- c(elevator_1[1], elevator_2[1], elevator_3[1])
                        direction[which.min(times)]
                        })
puzzle_5_sim_2_res <- mean(puzzle_5_sim_2)
print(puzzle_5_sim_2_res)

## [1] 0.6463
```

Hence the probability that first arriving elevator at Gamow's floow is going down is 0.6464.

Learning

The author provides code for a general case of M elevators. Looking at the solution, I have rewritten my initial somewhat clumsy code and generalized it for any M elevators

```
get_puzzle_15_prob <- function(M){
  n               <- 10000
  directions      <- matrix(sample(c(0, 1), n , TRUE) , nrow =  M, ncol = n)
  times           <- matrix(runif(n*M, 0, 6) , nrow =  M, ncol = n)
  p15_sim_results <- sapply(1:n,function(case){
    res           <- apply(cbind(directions[,case],times[,case]), 1,
                           function(case) waiting_time(case[1],case[2]))
    res[1,which.min(res[2,])]
      }
  )
  mean(p15_sim_results)
}
Ms                <- 2:10
puzzle_15_res     <- sapply(Ms,get_puzzle_15_prob)
puzzle_15_gen_res <- data.frame( M = Ms, probability= puzzle_15_res)
```



M	probability
2	0.7241
3	0.6427
4	0.5974
5	0.5590
6	0.5521
7	0.5294
8	0.5121
9	0.5079
10	0.5090

6 Steve's Elevator Problem

My attempt

For $k = 2$ riders, the closed form solution can be easily written as

$$\mu = \frac{9}{n^2} + \frac{14(n-3)}{n^2} + \frac{3(n-3)(n-4)}{n^2}$$

```
p5_closed_form <- function(n){9/n^2 + 14*(n-3)/n^2 + 3*(n-3)*(n-4)/n^2}
print(p5_closed_form(11))

## [1] 2.388
```

```
m       <- 1:15
n       <- 11
steve   <- n-2
cases   <- data.frame( n = n, m = m, stevesfloor = steve )
average <- apply(cases, 1, function(case){
             mean(replicate(10000,sum(unique(sample(1:case[1], case[2]-1, T)) < case[3])+1))
           })
puzzle_6_results <- cbind(cases, average)
```

n	m	stevesfloor	average
11	1	9	1.0000
11	2	9	1.7246
11	3	9	2.3867
11	4	9	3.0010
11	5	9	3.5381
11	6	9	4.0318
11	7	9	4.4813
11	8	9	4.8965
11	9	9	5.2756
11	10	9	5.6268
11	11	9	5.9019
11	12	9	6.1864
11	13	9	6.4577
11	14	9	6.6837
11	15	9	6.8814

Learning

The author gives a reference to the closed form solution to the above puzzle

$$\mu = 9 - 8\left(\frac{10}{11}\right)^k$$

7 The Pipe Smoker's Discovery

My attempt

```
puzzle_7           <- function(n){
  results          <- replicate(1000,{
    matches        <- c(n,n)
    while(TRUE){
      idx          <- ifelse(runif(1)>0.5,1,2)
      if(matches[idx]==0) break
      matches[idx] <- matches[idx]-1
    }
    2*n - sum(matches)
   })
  mean(results)
}
n_40               <- puzzle_7(40)
n_150              <- puzzle_7(150)
cat(n_40,n_150, "\n")

## 73.56 287.2
```

- The average number of matches that can be removed until one booklet is completely empty is 73.557
- The amount of floss in the other box is 12.752

Learning

The author introduces a variant of the puzzle - Banach Matchbox problem

> A mathematician who loved cigarettes (Banach smoked up to five packs a day and his early death at age 53 was, not surprisingly, of lung cancer) has two matchboxes, one in his left pocket and one in his right pocket, with each box initially containing N matches. The smoker selects a box at random each time he lights a new cigarette, and the problem is to calculate the probability that, when the smoker first discovers that the box he has just selected is empty, there are exactly r matches in the other box, where clearly $r = 0, 1, 2, \ldots, N$

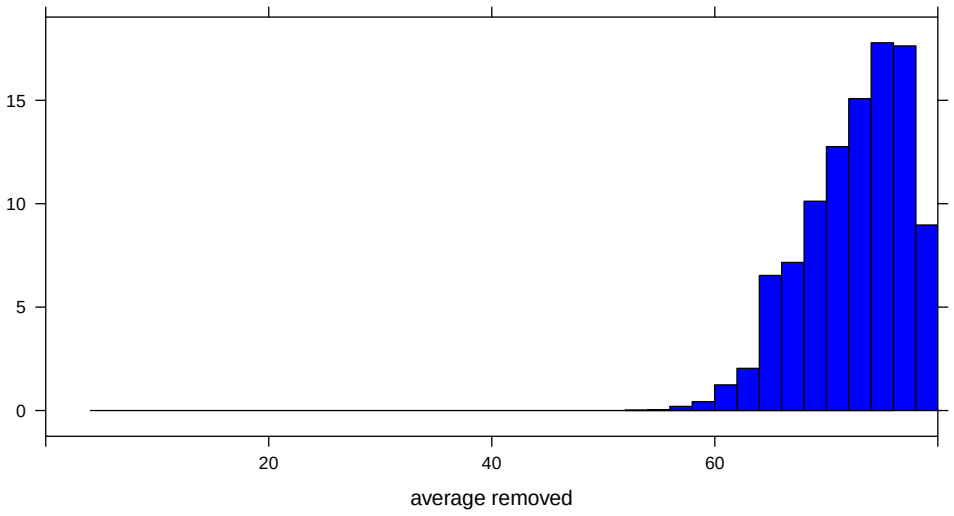
When I read the above variant, I realized that my code was actually not solving the original problem but solving Banach Matchbox problem. My code stops only when the person picks up a matchbox and discovers it as empty. The original puzzle is different. The count should stop when the matchbox becomes empty and not when the guy who picks it up realizes that it is empty.

Hence the actual code for the puzzle should be

```
puzzle_7           <- function(n){
  results          <- replicate(1000,{
    matches        <- c(n,n)
    while(TRUE){
      if(any(matches==0)) break
      idx          <- ifelse(runif(1)>0.5,1,2)
      matches[idx] <- matches[idx]-1
    }
    2*n - sum(matches)
   })
  mean(results)
}
n_40               <- puzzle_7(40)
n_150              <- puzzle_7(150)
cat(n_40,n_150, "\n")

## 73.23 286.2
```

- The average number of matches that can be removed until one booklet is completely empty is 73.228
- The amount of floss in the other box is 13.796

The author shows a histogram for the simulations and points at the peculiarities of the graph



The probabilities associated with average being less than 50 is 0 in the simulations as they are all rare events.

Closed form solution :

The author derives the closed form solution for the problem and shows the paradoxical nature of the problem.The probability that either of the boxes end up with k ft of floss is

$$P(k) = (1/2)^{2N-k-1} \frac{(2N-k-1)!}{(N-1)!(N-K)!}$$

and hence the average floss in the non-empty box is

$$\overline{k} = \sum_{k=1}^{N} kP(k) = \sum_{k=1}^{N} k(1/2)^{2N-k-1} \frac{(2N-k-1)!}{(N-1)!(N-K)!}$$

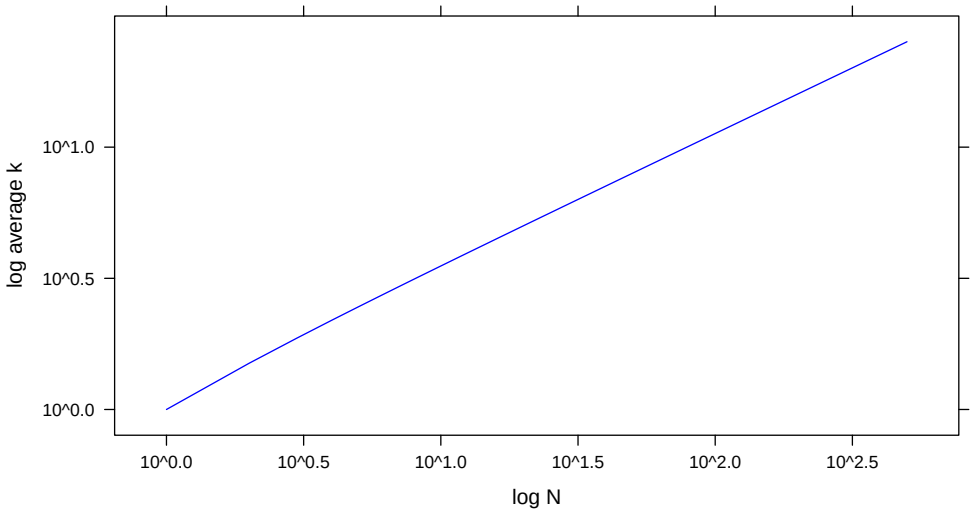
One can directly use the above formula to solve the puzzle

```
k           <- 1:40
N           <- 40
n_40_closed <- sum(k*(1/2)^(2*N-k-1) *(choose(2*N-k-1,N-k)))

k           <- 1:150
N           <- 150
n_150_closed <- sum(k*(1/2)^(2*N-k-1) *(choose(2*N-k-1,N-k)))
```

- The average number of matches that can be removed until one booklet is completely empty is 7.1142
- The amount of floss in the other box is 286.1917

One can even approximate the closed form solution as $\overline{k} = C\sqrt{N}$. This can be seen by drawing a log-log plot

```
Ns                 <- 1:500
p7_closedf_results <- sapply(Ns, function(N){
                  k <- 1:N
     sum(k*(1/2)^(2*N-k-1) *(choose(2*N-k-1,N-k)))
   }
  )
coefs <- coef(lm(log(p7_closedf_results)~log(Ns)))
C     <- coefs[1]
coefs[2]

## log(Ns)
##  0.5042
```



Hence an approximate solution is $\overline{k} = C\sqrt{N}$ where C is 0.0971.

The author also explains the paradox behind the puzzle, i.e. for any value of N, the most probable value of k is always less than the average value of k. The logic follows from the observation that for any N, $P(1) = P(2) < P(2) < P(4) \ldots$. Hence the dental floss problem is called a paradox.

8 A Toilet Paper Dilemma

My attempt

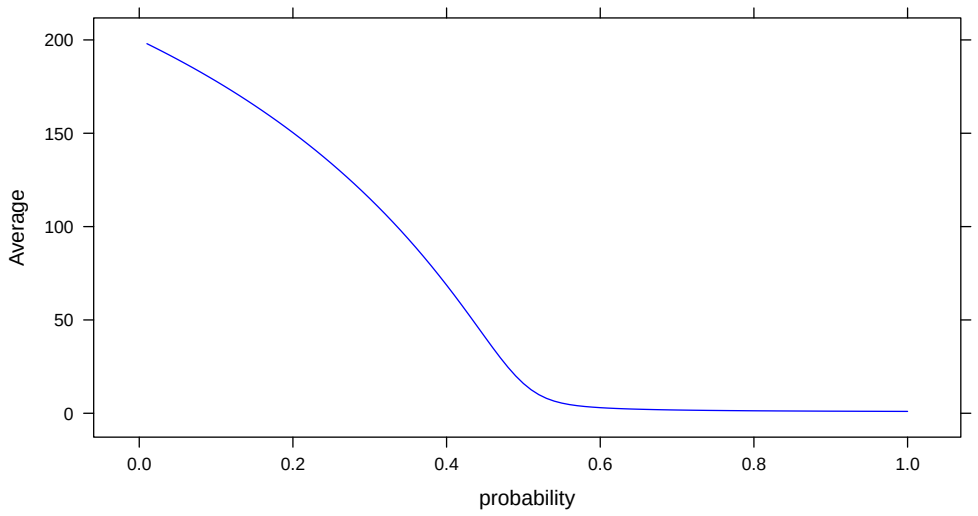
My attempt at solving this problem was disastrous. I ended up unnecessarily complicating the code by having a recursive function. The following is the code that I wrote which takes eternity for bigger m and n values

```
average_roll_left <- function(m,n, p){
  results  <- matrix(, ncol = 3 )[-1,]
  results  <- rbind(results, c(1,1,1))
  memoed   <- function(x,y) {
    sum(results[,1]==x & results[,2]==y) > 0
  }
M        <- function(m, n , p){
  if(memoed( m, n) ==TRUE) {
    cond <- results[,1]==m & results[,2]==n
    return(results[cond,3])
  }

  if(m == 0) {
   if(memoed(m,n)==FALSE) results  <<- rbind(results, c(0,n,n))
   return(n)
  }

  if(n == 0) {
   if(memoed(m,n)==FALSE) results  <<- rbind(results, c(m,0,m))
   return(m)
  }
  if(n > 0 & m == n) {
    temp <- M(n, n-1, p)
    if(memoed(m,n)==FALSE) results  <<- rbind(results, c(m,n,temp))
    return(temp)
  }

  if(m > n & n > 0 ) {
    temp     <- p*M(m - 1, n, p) + (1-p)*M(m, n - 1, p)
    if(memoed(m,n)==FALSE) results  <<- rbind(results, c(m,n,temp))
    return(temp)
  }
  }
M(m,m,p)
}
average_roll_left(10,10,0.5)

## [1] 3.524
```

Learning

The author's code is simple and elegant.

```
M              <- matrix(0,200,200)
prob           <- 0
k              <- 1
puzzle_8_results <- data.frame(prob = rep(0,100), value = rep(0,100))
for(k in 1:100){
  prob         <- prob + 1
  p            <- prob/100
  M[1, 1]      <- 1
  for(r in 2:200){
    M[1, r]    <- p*M[1, r-1] + (1 - p)*r
  }
  M[2, 2]      <- M[1,2]
  i            <- 2
  for(i in 2:199){
    for(j in (i+1):200){
      M[i, j] <- p*M[i, j - 1 ]  +(1 - p)*M[i - 1, j]
    }

    M[i+1, i+1]    <- M[i, i+1]
  }
  puzzle_8_results[k,1] <- p
  puzzle_8_results[k,2] <- M[200,200]
}
```



9 The Forgetful Burglar problem

My attempt

```
puzzle_9_trial   <- function(){
  loc            <- 0
  visited        <- c(loc)
  for(k in 1:7){
    jump         <- sample(c(-1,1),1)*sample(c(1,2),1)
    if((loc + jump) %in% visited){
        return(k)
    }else{
      loc        <- loc+jump
      visited    <- c(visited, loc)
    }
  }
  return(0)
}
puzzle_9_results_sim <- table(replicate(100000,puzzle_9_trial()))
puzzle_9_results_sim <- as.data.frame(cbind(puzzle_9_results_sim)/100000)[-1,,drop=FALSE]
colnames(puzzle_9_results_sim) <- "probability"
```

The following gives the probability of revisiting the same location after $k = 2 : 7$ steps. For $k = 1$, it is obvious that the probability is 0.

	probability
2	0.2524
3	0.2788
4	0.1926
5	0.1177
6	0.0693
7	0.0396

Learning

The author suggests that the generalized solution to this puzzle is still an open problem. However for the specific k=1 : 7, I guess one can write a difference equation and solve the probabilities recursively. The math has been worked out in the paper titled, "The case of the forgetful burglar", authored by Caxton Foster and Anatol Rapoport. The paper derives the probabilities for $k = 1 : 7$

10 The Umbrella Quandary

My attempt

CASE 1 : $x = 1, y = 1$

For the simple case of $x = 1, y = 1$, one can list down the state space. The following are the six transient states and one absorbing state

- Transient states A, B, C, D, E, F can be denoted as

$$(1, 1, \text{Home}), (2, 0, \text{Home}), (0, 2, \text{Home}), (1, 1, \text{Office}), (2, 0, \text{Office}), (0, 2, \text{Office})$$

  where the triple has the coordinates,(# of umbrellas at home ,# of umbrellas at office,current location).
- The absorbing state is the state when the man gets drenched. It is denoted by W

The transition matrix for the above states are

```
transition_matrix <- rbind(
      c(rep(0,3),0.5,0,0.5,0),c(rep(0,3),0.5,0.5,0,0),c(rep(0,3),0,0,0.5,0.5),
       c(0.5,0.5,0,rep(0,3),0),c(0,0.5,0,rep(0,3),0.5),c(0.5,0,0.5,rep(0,3),0),
       c(rep(0,6),1))
colnames(transition_matrix) <- c(LETTERS[1:6],"W")
rownames(transition_matrix) <- c(LETTERS[1:6],"W")
```

	A	B	C	D	E	F	W
A	0.0	0.0	0.0	0.5	0.0	0.5	0.0
B	0.0	0.0	0.0	0.5	0.5	0.0	0.0
C	0.0	0.0	0.0	0.0	0.0	0.5	0.5
D	0.5	0.5	0.0	0.0	0.0	0.0	0.0
E	0.0	0.5	0.0	0.0	0.0	0.0	0.5
F	0.5	0.0	0.5	0.0	0.0	0.0	0.0
W	0.0	0.0	0.0	0.0	0.0	0.0	1.0

For the above matrix, consider the submatrix $Q = [A : F, A : F]$, then the time to absorption from various starting states can be computed by the analytical formula $(I - Q)^{-1} \cdot \mathbf{I_6}$

```
average_steps <- (solve(diag(6)-transition_matrix[1:6,1:6])%*%rep(1,6))[1,1]-1
puzzle_10_closedform  <- average_steps
cat("On an average, the man will remain dry for ",average_steps," walks")

## On an average, the man will remain dry for  11  walks
```

CASE 2 : $x = 2, y = 2$

For this case, formulating the transition matrix is tedious. Hence resorted to simulation.

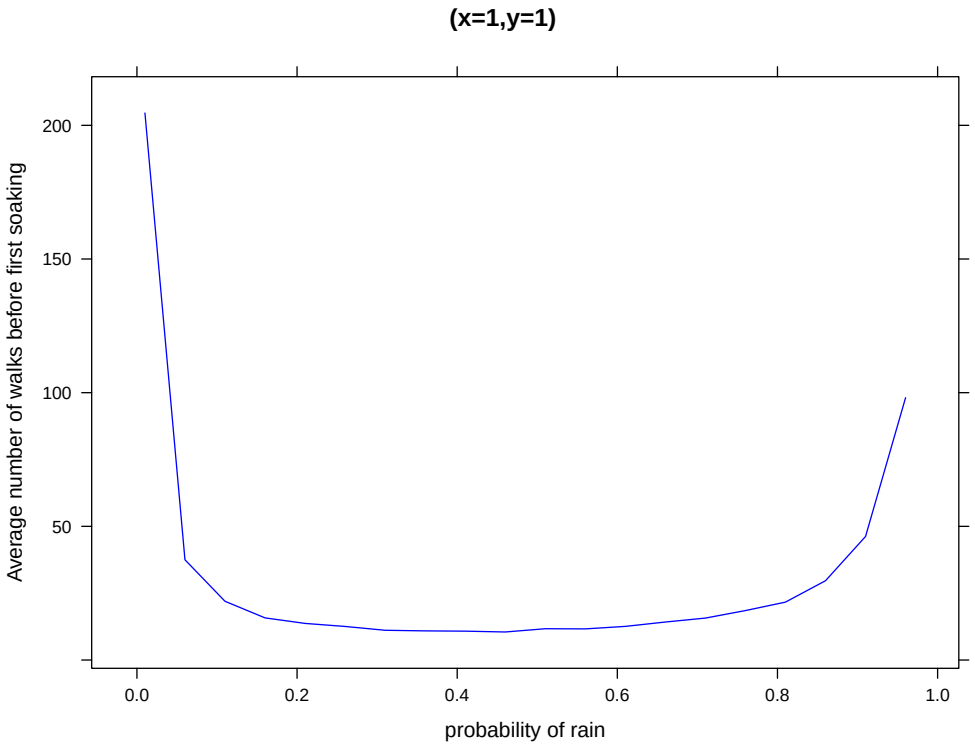
```
time_to_drench <- function(state,p){
  rep_p10       <- replicate(1000,{
  times         <- 0
  while(TRUE){
    outcome   <- ifelse(runif(1)>p, 1, 0)
    if(state[3]==1){
      state <- state + c(-outcome , +outcome , -1 )
    }else{
      state <- state + c(outcome ,-outcome , 1 )
    }
    if(state[1] < 0 | state[2] < 0) break
    times      <- times + 1
  }
  times
  }
)
mean(rep_p10)
}
puzzle_10_sim               <- time_to_drench(c(1,1,1),0.5)
puzzle_10_results           <- data.frame(x = 1,y=1,
                                          puzzle_10_closedform,
                                          puzzle_10_sim
                                          )
colnames(puzzle_10_results)  <- c("x","y", "closed form","simulation")
```

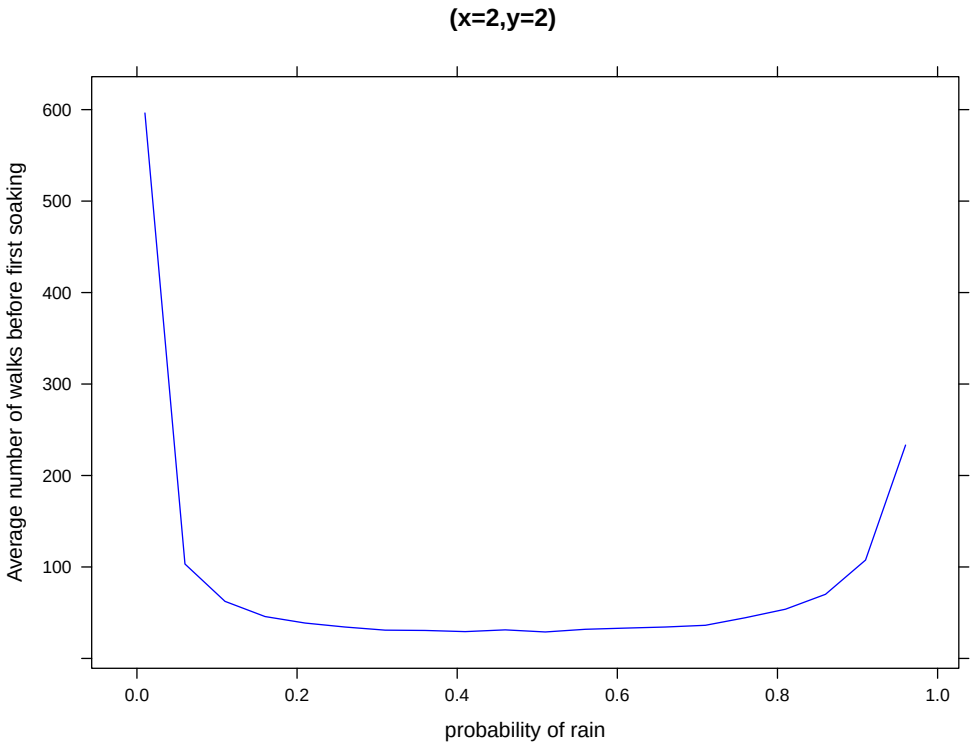
x	y	closed form	simulation
1	1	11	11

Learning

Nahin's code is a slightly longer, understandably, as the intention is to provide the reader with absolute clarity what the code does. The puzzle asks the reader to vary the probability of rain and compute the expected number of walks before soaking. For extreme situations $p = 0$(it never rains) and $p = 1$(it always rains), the man will never have a problem as he will never carry the umbrella or always carry the umbrella. Only when the probability is between 0 and 1, does the puzzle become analytically challengin. The following code computes the average steps for the two cases:

```
pval   <- seq(0.01,0.99, by=0.05)
case1  <- sapply(pval, function(z)time_to_drench(c(1,1,1),z))
case2  <- sapply(pval, function(z)time_to_drench(c(2,2,1),z))
```


(x=1,y=1)
Average number of walks before first soaking
200
150
100
50
0.0
0.2
0.4
0.6
0.8
1.0
probability of rain


(x=2,y=2)
Average number of walks before first soaking
600
500
400
300
200
100
0.0
0.2
0.4
0.6
0.8
1.0
probability of rain

11 The case of missing senator

My attempt

The puzzle boils down to figuring out the ways in which group-1(let's call it the minority group) ends up in over turning the proposal. Let x be the senators absent in group-1 and let y be the senators absent in group-2. Then for group-1 to win, the following must be satisfied

$$(49 - x) - (51 - y) \geq 1 \Rightarrow y - x \geq 3$$

Since $x + y = M$, where M is the total number of absentees, the above inequality becomes

$$y \geq \frac{M+3}{2}$$

Thus the constraint on y is $\lceil (M+3)/2) \rceil < y \leq M$, $M \geq 3$ Hence the solution boils down to assigning various probabilities depending on what value y takes.

```
puzzle_11_get_probability <- function(M){
  y  <- ceiling((M+3)/2):M
  sum((choose(51,y)*choose(49,(M-y)))/choose(100,M))
}
puzzle_11_closedform <- sapply(c(3,4,5),puzzle_11_get_probability)
```

Learning

After obtaining the closed form solution, I was not motivated to perform simulation for this puzzle. The author simulates the outcomes and estimates the probabilities. I rewrote the code in R

```
x                       <- rep(1,100)
puzzle_11_est_probability <- function(M,N){
 mean(replicate(N, {
    x[sample(1:100, M)] <- 0
    ifelse(sum(x[1:49]) - sum(x[50:100]) >= 1, 1, 0)
 }))
}
puzzle_11_sim               <- sapply(c(3,4,5),puzzle_11_est_probability, N=100000)
puzzle_11_results           <- data.frame(M = 3:5,puzzle_11_closedform,puzzle_11_sim)
colnames(puzzle_11_results)  <- c("M","closed form", "simulation")
```

M	closed form	simulation
3	0.12879	0.12969
4	0.06373	0.06323
5	0.19385	0.19480

12 How many runners in a marathon ?

My attempt

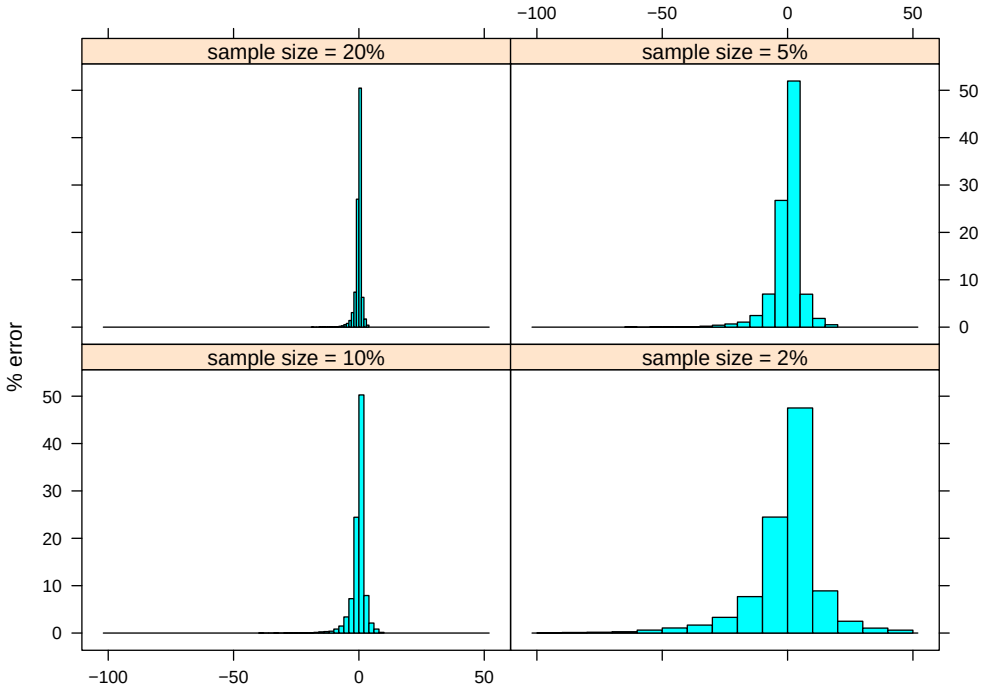
```
props       <- c(2, 5, 10, 20) / 100
M           <- 10000
trial       <- function(p){
    sim     <- replicate(M, {
       N      <- runif(1, 100, 1000)
       n      <- floor(p*N)
       N_est  <- max(sample(1:N,n ))*(n+1)/n  - 1
       (100*(N_est/N - 1))
     })
  c(mean(sim),sd(sim))
}
puzzle_12_results            <- data.frame(props, t(sapply(props, trial)))
colnames(puzzle_12_results)  <- c("sample size % ","avg of error %","sd of error%")
```

sample size %	avg of error %	sd of error%
0.02	-0.01	14.01
0.05	-0.12	6.00
0.10	-0.10	2.95
0.20	-0.12	1.39

Learning

The author illustrates the histograms of various trials and shows that standard deviation goes down as the sample size increases

```
props       <- c(2, 5, 10, 20) / 100
M           <- 10000
trial       <- function(p){
    sim     <- replicate(M, {
       N      <- runif(1, 100, 1000)
       n      <- floor(p*N)
       N_est  <- max(sample(1:N,n ))*(n+1)/n  - 1
       (100*(N_est/N - 1))
     })
    sim
}
puzzle_12_sim            <- data.frame(val = c(sapply(props, trial)),
                                       rep(paste0("sample size = ",props*100, "%"),each = 10000))
colnames(puzzle_12_sim)  <- c("pct_error","sample_proportion")
```



13 A Police Patrol Problem

My attempt

For me, this puzzle has been the most challenging puzzle of the entire lot. Firstly, the way you simulate the paths is not straightforward. One needs to think through an efficient way to simulate and store it in a datastructure

Capture all scenarios in a data frame for easy lookup

```
scenarios      <- matrix(,1,4)[-1,]
for(patrol in 1:2){
  for(accident in 1:2){
    for(direction in 0:1){
      for(divide in 0:1){
        scenarios <- rbind(scenarios, c(patrol, accident, direction, divide))
      }
    }
  }
}
colnames(scenarios) <- c("patrol","accident", "direction", "divide")
```

Function to compute the minimum distance for a specific realization

```
distance_to_accident <- function(patrol_lane, patrol_loc, direction, accident_lane,
                                 accident_loc, divide){
 m                   <- length(patrol_lane)
 result              <- numeric(m)
 for(i in 1:m){
   x                 <- accident_loc
   y                 <- patrol_loc[i]
   situation         <- which(scenarios[,1] == patrol_lane[i] &
                         scenarios[,2] == accident_lane &
                         scenarios[,3] == direction[i] &
                         scenarios[,4] == divide)
    result[i]        <- switch(situation,
                      y-x, 2+x-y, x-y, x-y, y-x, 2-x-y,
                      x-y, 2-x-y, x-y, x+y, y-x, x+y,
                      x-y , 2-x+y, y-x, y-x)
   }
  return(min(result))
 }
```

Function that generates samples for various situations

```
samples                    <- function(m, n,fixed=FALSE){
   realizations            <- cbind( matrix( sample(c(1, 2),(n + 1) * m, T),
                                            nrow = m, ncol = n + 1),
                              matrix( runif((n + 1) * m),nrow = m, ncol = n + 1),
                              matrix( sample(c(0, 1),m, T), nrow = m, ncol = 1))
   colnames(realizations) <- c( paste0("patrol_lane", 1:n),
                                "accident_lane", paste0("patrol_loc", 1:n),
                                      "accident_loc", "divide")
   if(fixed == TRUE)  realizations[,"patrol_loc1"] <- 0.5
   realizations            <- as.data.frame(realizations)
   directions              <- matrix(0, nrow = m, ncol = n)
   colnames(directions)    <- paste0("dir",1:n)
   for(i in 1:n){
    cond                   <- realizations[,i]==1 & realizations$accident_lane ==1
    directions[cond,i]     <- ifelse(realizations[cond,]$accident_loc >
                                     realizations[cond,(n+1+i)], 1, 0)

    cond                   <- realizations[,i]==2 & realizations$accident_lane ==2
    directions[cond,i]     <- ifelse(realizations[cond,]$accident_loc >
                                     realizations[cond,(n+1+i)], 0, 1)

    cond                   <- realizations[,i]==1 & realizations$accident_lane ==2
    directions[cond,i]     <- ifelse(realizations[cond,]$accident_loc >
                                     realizations[cond,(n+1+i)], 1, 0)

    cond                   <- realizations[,i]==2 & realizations$accident_lane ==1
    directions[cond,i]     <- ifelse(realizations[cond,]$accident_loc >
                                     realizations[cond,(n+1+i)], 0, 1)
   }
    cbind(realizations[,c(1:n,(n+2):(2*n+1))],directions, realizations[,c(n+1,2*n+2,2*n+3)])
  }
```

Function to compare the various configurations

```
get_average_response <- function(n, fixed=FALSE){
  test                 <- samples(1000,n, fixed)
  test$dist            <- apply(test, 1, function(z)
                          distance_to_accident(z[1:n], z[(n+1):(2*n)],
                                   z[(2*n+1) : (3*n)],z[3*n+1],
                                   z[3*n+2],z[3*n+3]))
  with(test, tapply(dist,divide,mean))
}
```

Comparing various configurations

```
puzzle_13_results <- c(get_average_response(1,TRUE),get_average_response(1),
                       get_average_response(2))
puzzle_13_results <- data.frame(scenarios = c("a-1","a-2","b-1","b-2","c-1","c-2"),
                                average = puzzle_13_results)
```

scenarios	average
a-1	0.2435
a-2	1.0083
b-1	0.3275
b-2	1.0339
c-1	0.2036
c-2	0.6540

From the above table, one can see that scenario c-1 is the best performing one

Learning

The author gives a detailed flowchart to write the code necessary to solve the problem. He also provides closed form solution to one of the cases.

$$\begin{aligned}\text{grassy median} &= \frac{1}{2n+1}\\ \text{concrete median} &= \frac{2}{n+1}\end{aligned}$$

Hence as $n \to \infty$, the ratio of response time for concrete median to grassy median should tend to 4. One can verify this using simulation

```
puzzle_13_ratio      <- t(sapply(c(1,2,3,4,9), get_average_response))
puzzle_13_ration_res <- data.frame(cars  = c(1,2,3,4,9),
                                   ratio = puzzle_13_ratio[,2]/puzzle_13_ratio[,1])
```

cars	ratio
1	2.9268
2	3.0440
3	3.1061
4	3.2713
9	3.6636

14 Parrondo's Paradox

My attempt

Simulate Game A

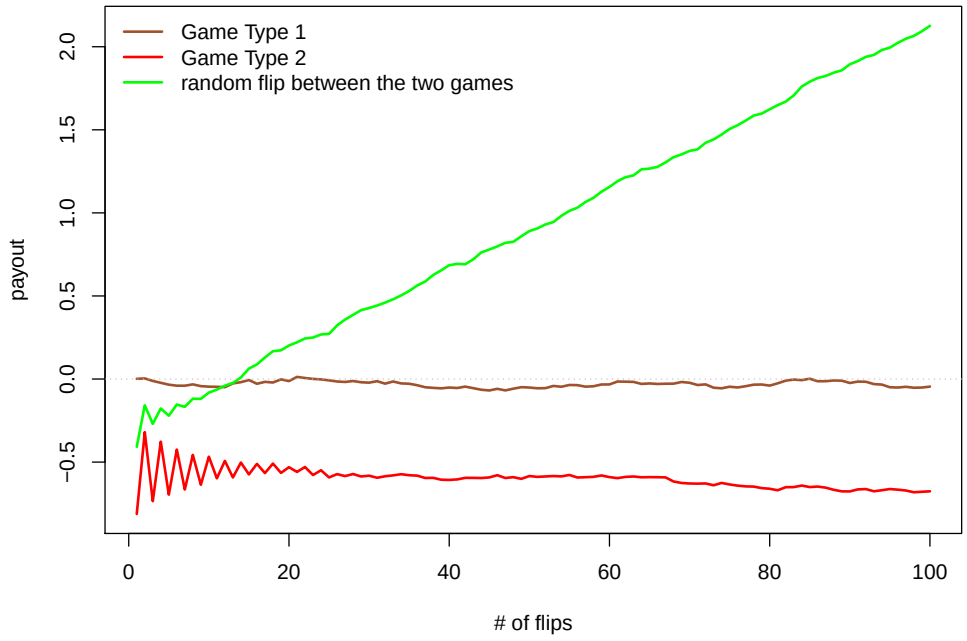
```
epsilon        <- 1e-3
p1             <- 1/10-epsilon
p2             <- 3/4-epsilon
p3             <- 1/2 -epsilon
N              <- 101
M              <- 10000
results        <- matrix(0, nrow = M, ncol = N)
for(j in seq_len(M)){
  score        <- numeric(N)
  i            <- 1
  score[i]     <- 0
  for(i in seq_len(N)[-1]){
    outcome    <- ifelse(runif(1)<p3, 1, -1)
    score[i]   <- score[i-1] + outcome
  }
  results[j,]  <- score
}
results_game1  <- colMeans(results)
```

Simulate Game B

```
results         <- matrix(0, nrow = M, ncol = N)
for(j in seq_len(M)){
  score         <- numeric(N)
  i             <- 1
  score[i]      <- 0
  for(i in seq_len(N)[-1]){
    if(score[i-1]%%3==0){
      outcome   <- ifelse(runif(1)<p1, 1, -1)
    }else{
      outcome   <- ifelse(runif(1)<p2, 1, -1)
    }
    score[i]    <- score[i-1] + outcome
  }
  results[j,]   <- score
}
results_game2   <- colMeans(results)
```

Simulate flipping between Game A and Game B

```
epsilon          <- 1e-3
p1               <- 1/10-epsilon
p2               <- 3/4-epsilon
p3               <- 1/2 -epsilon

results          <- matrix(0, nrow = M, ncol = N)
for(j in seq_len(M)){
  score          <- numeric(N)
  i              <- 1
  score[i]       <- 0
  for(i in seq_len(N)[-1]){
    if(runif(1) > 0.5){
      if(score[i-1]%%3==0){
        outcome  <- ifelse(runif(1)<p1, 1, -1)
      }else{
        outcome  <- ifelse(runif(1)<p2, 1, -1)
      }
    }else{
      outcome    <- ifelse(runif(1)<p3, 1, -1)
    }
    score[i]     <- score[i-1] + outcome
  }
  results[j,]    <- score
}
results_game12 <- colMeans(results)
```



Learning

The logic of the code was very much identical to what I have coded. The notes suggest some interesting papers that explore the math behind the paradox.

- Brownian motion and gambling: from ratchets to paradoxical games
- Simple games to illustrate Parrondo's paradox

15 How Long Is the Wait to Get the Potato Salad ?

My attempt

This is a classic M/M/1 problem from the queueing theory. The questions in this puzzle are

1. What is the average total time at the deli counter for the customers (total time is the sum of the waiting time and the service time) ?
2. What is the maximum total time experienced by the unluckiest of the customers ?
3. What is the average length of the customer waiting queue?
4. What is the maximum length of the customer waiting queue?
5. What happens to the answers to the first four questions if a second, equally skilled deli clerk is hired?
6. What is the fraction of the work day that the clerks are idle ?

```
Queue <- setRefClass(Class = "Queue",
                     fields = list(
                       name = "character",
                       data = "list"
                     ),
                     methods = list(
                       size = function() {
                         return(length(data))
                       },
                       push = function(item) {
                         data[[size()+1]] <<- item
                       },
                       pop = function() {
                         if (size() == 0) stop("queue is empty!")
                         value <- data[[1]]
                         data[[1]] <<- NULL
                         value
                       },
                       poll = function() {
                         if (size() == 0) return(NULL)
                         else pop()
                       },
                       peek = function(pos = c(1)) {
                         if (size() < max(pos)) return(NULL)
                         if (length(pos) == 1) return(data[[pos]])
                         else return(data[pos])
                       },
                       initialize=function(...) {
                         callSuper(...)
                         .self
                       }
                     ))
```

Simulate M/M/1

```
simulate_m_m_1     <- function(lambda, mu, duration){
  t                <- 0
  wq               <- Queue$new()
  wq$name          <- "deli queue"
  servq            <- Queue$new()
  servq$name       <- "servicing queue"
  customer         <- 0
  max_queue_size   <- 0
  t1               <- rexp(1,rate=lambda)
  customer         <- customer + 1
  servq$push(c(customer,t1,t1,0))
  t                <- t1
  result           <- data.frame(customer=numeric(), entry_time = numeric(),
                                 serv_start = numeric() , serv_end = numeric())
  while( t < duration) {
    if( servq$size() >0  ) {
      t1           <- rexp(1,rate = lambda + mu)
      p            <- runif(1)
      if(p < lambda/(lambda + mu)){ #arrival
        customer   <- customer + 1
        wq$push(c(customer, t + t1, 0 , 0))
    } else {
        serviced         <- servq$pop()
        serviced[4]      <- t+t1
        names(serviced)  <- colnames(result)
        result           <- rbind(result,as.list(serviced))
        if(wq$size() >0){
          temp           <- wq$pop()
          temp[3]        <- t+t1
          servq$push(temp)
          }
      }
    } else {
      t1                 <- rexp(1,rate=lambda)
      customer           <- customer + 1
      servq$push(c(customer,t+t1,t+t1,0))
    }
    if(max_queue_size < wq$size()) max_queue_size <- wq$size()
    t                    <- t + t1
  }
  avg_total_time        <- mean(result$serv_end-result$entry_time)
  max_time              <- max(result$serv_end-result$entry_time)
```

```
  avg_queue_length      <- sum(result$serv_start-result$entry_time)/36000
  idle_time             <- (1- sum(result$serv_end-result$serv_start)/36000)*100
  c(avg_total_time,max_time,avg_queue_length, max_queue_size,idle_time)
}
```

Single Server $\lambda = 30, \mu = 40$

```
duration                <- 36000
lambda                  <- 30/3600
mu                      <- 40/3600
puzzle_15_results       <- replicate(5,simulate_m_m_1(lambda,mu,duration))
puzzle_15_results_one_s <- matrix(round((t(puzzle_15_results)),2),nrow=5)
colnames(puzzle_15_results_one_s) <- c("Average total time(sec)",
                                       "Max total time(sec)",
                                       "Average Q length",
                                       "Maximum Q length",
                                       "Idle time(percent)")
rownames(puzzle_15_results_one_s) <-NULL
puzzle_15_xtable          <- xtable(puzzle_15_results_one_s,digits=2)
align( puzzle_15_xtable ) <- c( 'l|p{1in}|p{0.75in}|p{0.75in}|p{0.75in}|p{0.75in}' )
```

	Average total time(sec)	Max total time(sec)	Average Q length	Maximum Q length	Idle time(percent)
1	226.44	776.65	1.14	8.00	31.18
2	275.90	1228.21	1.40	10.00	28.83
3	356.77	1254.41	2.26	10.00	19.22
4	291.77	1135.25	1.79	11.00	26.65
5	526.21	2005.42	3.95	20.00	19.96

Computing the average value from 100 simulations to theoretical values

```
duration                <- 36000
lambda                  <- 30/3600
mu                      <- 40/3600
puzzle_15_results       <- replicate(100,simulate_m_m_1(lambda,mu,duration))
puzzle_15_results_one_s <- matrix(round(colMeans(t(puzzle_15_results)),2),nrow=5)
rownames(puzzle_15_results_one_s) <- c("Average total time(sec)",
                                       "Max total time(sec)",
                                       "Average Q length",
                                       "Maximum Q length",
                                       "Idle time(percent)")
rho                       <- lambda/mu
L                         <- rho/(1-rho)
L_q                       <- rho^2/(1-rho)
W                         <- L / lambda
W_q                       <- L_q / lambda
p_0                       <- (1-rho)*100
puzzle_15_results_one_s  <- cbind(puzzle_15_results_one_s,c(W,NA,L_q,NA,p_0))
colnames(puzzle_15_results_one_s) <- c("estimated average","closed form")
```

	estimated average	closed form
Average total time(sec)	340.40	360.00
Max total time(sec)	1316.97	
Average Q length	2.08	2.25
Maximum Q length	12.43	
Idle time(percent)	25.96	25.00

Single Server $\lambda = 30, \mu = 25$

```
duration                <- 36000
lambda                  <- 30/3600
mu                      <- 25/3600
puzzle_15_results       <- replicate(5,simulate_m_m_1(lambda,mu,duration))
puzzle_15_results_one_s <- matrix(round((t(puzzle_15_results)),2),nrow=5)
colnames(puzzle_15_results_one_s) <- c("Average total time(sec)",
                                       "Max Total time(sec)",
                                       "Average Queue Length",
                                       "Maximum Queue Length",
                                       "Idle time(percent)")
rownames(puzzle_15_results_one_s) <-NULL
puzzle_15_xtable          <- xtable(puzzle_15_results_one_s,digits=2)
align( puzzle_15_xtable ) <- c( 'l|p{1in}|p{0.75in}|p{0.75in}|p{0.75in}|p{0.75in}' )
```

	Average total time(sec)	Max Total time(sec)	Average Queue Length	Maximum Queue Length	Idle time(percent)
1	2212.74	5698.16	14.12	42.00	6.29
2	4443.20	6749.52	29.01	59.00	2.17
3	1624.39	3488.01	10.75	28.00	1.83
4	2929.41	5645.22	20.50	41.00	1.41
5	2409.19	6204.36	16.61	60.00	0.59

Simulate M/M/2

```
simulate_m_m_2     <- function(lambda, mu, duration){
  t                <- 0
  wq               <- Queue$new()
  wq$name          <- "deli queue"
  servq1           <- Queue$new()
  servq1$name      <- "servicing queue1"
  servq2           <- Queue$new()
  servq2$name      <- "servicing queue2"
  customer         <- 0
  max_queue_size   <- 0
  t1               <- rexp(1,rate=lambda)
  customer         <- customer + 1
  servq1$push(c(customer,t1,t1,0))
  t                <- t1
  result           <- data.frame(customer=numeric(), entry_time = numeric(),
                                 serv_start = numeric() , serv_end = numeric(),serv_no=numeric())
  while( t < duration) {

    if( servq1$size() > 0  | servq2$size() > 0  )  {
      t1           <- rexp(1, rate = lambda + 2*mu)
      p            <- runif(1)
      if(p < lambda/(lambda + 2*mu)){ #arrival
        customer   <- customer + 1
        if(servq1$size()==0){
          servq1$push(c(customer, t + t1, t+t1 , 0))
        }else{
          if(servq2$size()==0){
            servq2$push(c(customer, t + t1, t+t1 , 0))
          }else{
            wq$push(c(customer, t + t1, 0 , 0))
          }
        }
    } else {
        if(servq1$size()>0){
        serviced         <- servq1$pop()
        serviced[4]      <- t+t1
        serviced[5]      <- 1
        names(serviced)  <- colnames(result)
        result           <- rbind(result,as.list(serviced))

        }else{
          if(servq2$size()>0){
```

```
        serviced          <- servq2$pop()
        serviced[4]       <- t+t1
        serviced[5]       <- 2
        names(serviced)   <- colnames(result)
        result            <- rbind(result,as.list(serviced))
         }
       }
       if(wq$size() >0){
         temp           <- wq$pop()
         temp[3]        <- t+t1
         if(servq1$size() == 0){servq1$push(temp)}
         else{servq2$push(temp)}
         }
     }
   } else {
     t1                 <- rexp(1,rate=lambda)
     customer           <- customer + 1
     servq1$push(c(customer,t+t1,t+t1,0))
   }
   if(max_queue_size < wq$size()) max_queue_size <- wq$size()
   t                    <- t + t1
 }
 avg_total_time        <- mean(result$serv_end-result$entry_time)
 max_time              <- max(result$serv_end-result$entry_time)
 avg_queue_length      <- sum(result$serv_start-result$entry_time)/36000
 temp                  <- subset(result, serv_no==1)
 idle_time_1           <- (1- sum(temp$serv_end-temp$serv_start)/36000)*100
 temp                  <- subset(result, serv_no==2)
 idle_time_2           <- (1- sum(temp$serv_end-temp$serv_start)/36000)*100
 c(avg_total_time,max_time,avg_queue_length, max_queue_size,idle_time_1,idle_time_2)
}
```

@ Two Servers $\lambda = 30, \mu = 40$

```
duration                <- 36000
lambda                  <- 30/3600
mu                      <- 40/3600
puzzle_15_results       <- replicate(5,simulate_m_m_2(lambda,mu,duration))
puzzle_15_results_two_s <- matrix(round((t(puzzle_15_results)),2),nrow=5)
colnames(puzzle_15_results_two_s) <- c("Average total time(sec)",
                                       "Max Total time(sec)",
                                       "Average Queue Length",
                                       "Maximum Queue Length",
                                       "Server 1 Idle time(percent)",
```

```
                                        "Server 2 Idle time(percent)")
rownames(puzzle_15_results_two_s) <-NULL
puzzle_15_xtable            <- xtable(puzzle_15_results_two_s,digits=2)
align( puzzle_15_xtable )<- c( 'l|p{1in}|p{0.75in}|p{0.75in}|p{0.75in}|p{0.75in}|p{0.75in}' )
```

	Average total time(sec)	Max Total time(sec)	Average Queue Length	Maximum Queue Length	Server 1 Idle time(percent)	Server 2 Idle time(percent)
1	62.87	640.61	0.05	4.00	72.72	81.64
2	68.53	709.27	0.04	3.00	71.18	81.97
3	70.86	876.73	0.09	5.00	71.33	81.06
4	72.85	653.43	0.09	4.00	68.53	78.34
5	71.01	1038.37	0.10	4.00	67.10	78.06

Computing the average value from 100 simulations to theoretical values

```
duration                <- 36000
lambda                  <- 30/3600
mu                      <- 40/3600
puzzle_15_results       <- replicate(100,simulate_m_m_2(lambda,mu,duration))
puzzle_15_results_two_s <- matrix(round(colMeans(t(puzzle_15_results)),2),nrow=6)
rownames(puzzle_15_results_two_s) <- c("Average total time(sec)",
                                       "Max Total time(sec)",
                                       "Average Queue Length",
                                       "Maximum Queue Length",
                                       "Server 1 Idle time(percent)",
                                       "Server 2 Idle time(percent)")
r                         <- lambda/mu
rho                       <- lambda/(2*mu)
p_0                       <- (r^2/(2*(1-rho)) + 1 + r)^(-1)
L_q                       <- (r^2*rho/(2*(1-rho)^2))*p_0
W_q                       <- L_q / lambda
W                         <- 1/mu + (r^2/(2*2*mu*(1-rho)^2))*p_0
puzzle_15_results_two_s  <- cbind(puzzle_15_results_two_s,c(W,NA,L_q,NA,NA,NA))
colnames(puzzle_15_results_two_s) <- c("estimated average","closed form")
```

	estimated average	closed form
Average total time(sec)	71.99	104.73
Max Total time(sec)	753.88	
Average Queue Length	0.09	0.12
Maximum Queue Length	3.93	
Server 1 Idle time(percent)	68.73	
Server 2 Idle time(percent)	79.50	

Two Servers $\lambda = 30, \mu = 25$

```
duration                <- 36000
lambda                  <- 30/3600
mu                      <- 25/3600
puzzle_15_results       <- replicate(5,simulate_m_m_2(lambda,mu,duration))
puzzle_15_results_two_s <- matrix(round((t(puzzle_15_results)),2),nrow=5)
colnames(puzzle_15_results_two_s) <- c("Average total time(sec)",
                                       "Max Total time(sec)",
                                       "Average Queue Length",
                                       "Maximum Queue Length",
                                       "Server 1 Idle time(percent)",
                                       "Server 2 Idle time(percent)")
rownames(puzzle_15_results_two_s) <-NULL
puzzle_15_xtable            <- xtable(puzzle_15_results_two_s,digits=2)
align( puzzle_15_xtable )<- c( 'l|p{1in}|p{0.75in}|p{0.75in}|p{0.75in}|p{0.75in}|p{0.75in}' )
```

	Average total time(sec)	Max Total time(sec)	Average Queue Length	Maximum Queue Length	Server 1 Idle time(percent)	Server 2 Idle time(percent)
1	140.66	1511.12	0.25	5.00	52.08	58.46
2	157.57	1752.96	0.40	8.00	54.60	60.60
3	184.04	2119.06	0.42	6.00	46.60	52.52
4	169.54	2955.12	0.55	11.00	49.76	58.88
5	134.17	1132.38	0.20	4.00	56.86	65.01

Computing the average value from 100 simulations to theoretical values

```
duration                <- 36000
lambda                  <- 30/3600
mu                      <- 25/3600
puzzle_15_results       <- replicate(100,simulate_m_m_2(lambda,mu,duration))
puzzle_15_results_two_s <- matrix(round(colMeans(t(puzzle_15_results)),2),nrow=6)
rownames(puzzle_15_results_two_s) <- c("Average total time(sec)",
                                       "Max Total time(sec)",
                                       "Average Queue Length",
                                       "Maximum Queue Length",
                                       "Server 1 Idle time(percent)",
                                       "Server 2 Idle time(percent)")
r                         <- lambda/mu
rho                       <- lambda/(2*mu)
p_0                       <- (r^2/(2*(1-rho)) + 1 + r)^(-1)
L_q                       <- (r^2*rho/(2*(1-rho)^2))*p_0
W_q                       <- L_q / lambda
W                         <- 1/mu + (r^2/(2*2*mu*(1-rho)^2))*p_0
puzzle_15_results_two_s  <- cbind(puzzle_15_results_two_s,c(W,NA,L_q,NA,NA,NA))
colnames(puzzle_15_results_two_s) <- c("estimated average","closed form")
```

	estimated average	closed form
Average total time(sec)	177.90	225.00
Max Total time(sec)	2358.53	
Average Queue Length	0.54	0.67
Maximum Queue Length	7.41	
Server 1 Idle time(percent)	48.69	
Server 2 Idle time(percent)	55.97	

16 The Appeals Court Paradox

My attempt

```
probs    <- c(0.95, 0.95, 0.9, 0.9, 0.8)
(1 - mean(replicate(1000000, sum(rbinom(5, 1, prob = probs)) >= 3))) * 100

## [1] 0.7006

(1 - mean(replicate(1000000, sum(rbinom(5, 1, prob = probs)[c(1, 2, 3, 4, 1)]) >= 3))) * 100

## [1] 1.218
```

Learning

The author computes the failure probability directly. Here is the `R` code for it:

```
probs    <- c(0.95, 0.95, 0.9, 0.9, 0.8)
mean(replicate(1000000, sum(1- rbinom(5, 1, prob = probs)) > 2)) * 100

## [1] 0.7123

mean(replicate(1000000, sum(1- rbinom(5, 1, prob = probs)[c(1, 2, 3, 4, 1)]) > 2 )) * 100

## [1] 1.193
```

It is truly paradoxical.

> If the worst judge follows the lead of the best judge, then we have an increased (almost doubled) probability that the court erors! What happens to the concept of setting a good example? Most people are suprised by the result.

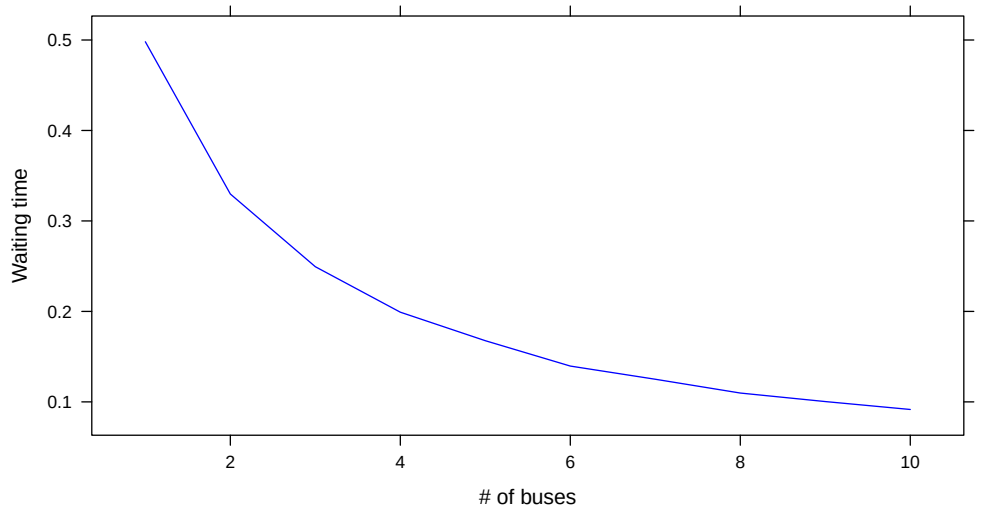
17 Waiting for Buses

My attempt

```
bus_waiting_time <- function(n){
  mean( replicate(10000,{
  timings       <- c(0,1)
  if(n > 1){
    for(bus in 2:n){
       x        <- runif(1)
       timings   <- c(timings,x)
    }
  }
  timings  <- timings[order(timings,decreasing = FALSE)]
  arrival  <- runif(1)
  idx      <- which(timings>=arrival)[1]
  timings[idx]-arrival
  }))
}
puzzle_17_results_sim <- sapply(1:10,bus_waiting_time)
puzzle_17_results_sim <- as.data.frame(puzzle_17_results_sim)
colnames(puzzle_17_results_sim) <- "waiting time"
```

The following gives the values of the waiting time for different number of buses :

	waiting time
1	0.50
2	0.33
3	0.25
4	0.20
5	0.17
6	0.14
7	0.13
8	0.11
9	0.10
10	0.09



One can guess that the relationship as

$$E(W_n) = \frac{1}{n+1}$$

Learning

The author provides a closed form solution for the problem for $n = 2$. Let's say that the bus arrives at x time, the probability of the rider arriving before x is x and the probability of the rider arriving after x is $1 - x$. The waiting time for the former case is $x/2$ and the later case is $(1 - x)/2$

$$E[W|x] = x/2 \cdot x + (1 - x)/2 \cdot (1 - x)$$

This implies

$$E[W] = \int_0^1 x/2 \cdot x + (1 - x)/2 \cdot (1 - x)dx = 1/3$$

18 Waiting for Stoplights

My attempt

In the first go, I had tried coming up with a simple solution based on combinatorics. I found it easier to visualize the puzzle where I am standing at $(0,0)$ and the objective is to reach (m,m). I would not stop anywhere until I hit $x = m$ or $y = m$, after which I need to wait if there is an appropriate signal. If you consider the $(m-1) \times (m-1)$ block and then compute the probabilities of hitting the $x = (m-1)$ or $y = (m-1)$, then these probabilities can be used to compute the expected waiting times.

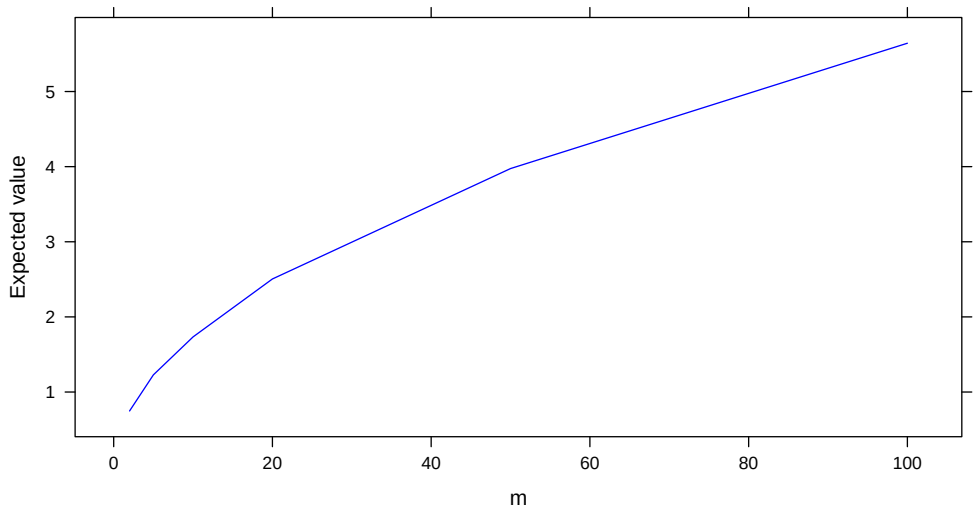
$$E(\text{wait}) = 2 \cdot \left(\sum_{j=0}^{m-1} \binom{m-1+j}{j} \cdot (1/2)^{m-1} \cdot (1/2)^{j} \cdot (1/2) \cdot (m-i)/2 \right)$$

```
closed_form <- function(m){
  idx       <- 0:(m-1)
  sum( choose(m - 1 + idx, idx) *  (1/2)^(m-1)* 1/2^(idx) *((m):1)*1/2 * 1/2) *2
}
puzzle_18_closedform <- sapply(c(2,5,10,20,50,100), closed_form)
```

Monte carlo method: Another way to solve this problem is to simulate paths

```
get_to_destination <- function(m){
  location         <- c(0,0)
  waiting          <- 0
  while(location[1]!=m & location[2]!=m ){
    direction      <- ifelse(runif(1)>0.5,1,0)
    if(direction==1) {
        location   <- location + c(1,0)
      }else{
        location   <- location + c(0,1)
      }
  }
  remaining        <- ifelse(location[1]==m,m-location[2],m-location[1])
  remaining/2
}
ms                 <- c(2,5,10,20,50,100)
n                  <- length(ms)
p18_results_1      <- numeric(n)
for(i in seq_len(n)){
  p18_results_1[i] <-   mean(replicate(5000,get_to_destination(ms[i])))
}
puzzle_18_results            <- data.frame(puzzle_18_closedform, p18_results_1)
colnames(puzzle_18_results)  <- c("closed form","simulation")
rownames(puzzle_18_results)  <- ms
```

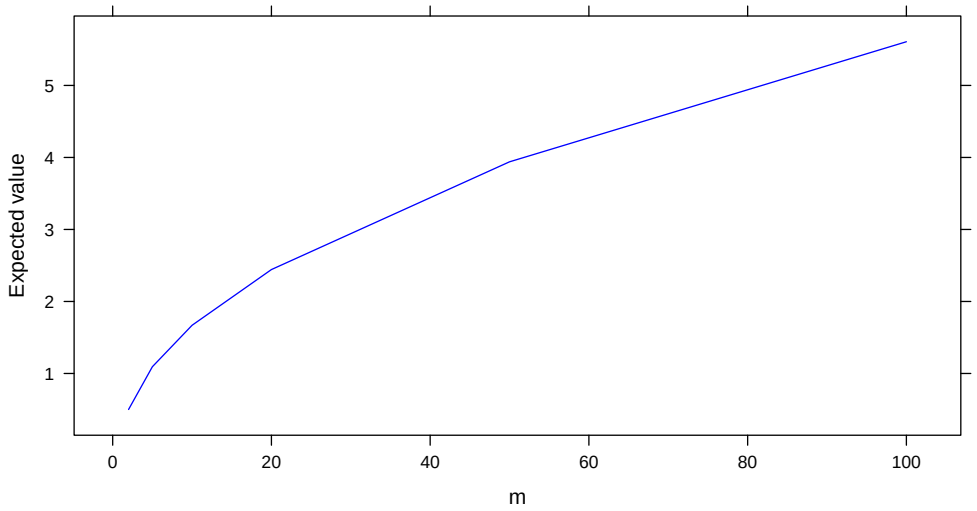
	closed form	simulation
2	0.75000	0.74780
5	1.23047	1.22820
10	1.76197	1.73360
20	2.50741	2.50500
50	3.97946	3.97460
100	5.63485	5.64270



Learning

Generating function approach

```
gen_fun_approach_p18 <- function(m){
  p                  <- matrix(0 , nrow= m, ncol = m)
  p[m,]              <- 0.5*((m-1):0)
  p[,m] <- 0.5*((m-1):0)
  for(i in (m-1):1) {
    for(j in (m-1):1) {
        p[i,j]      <- (p[i+1,j]+p[i,j+1])*0.5
  }
  }
  p[1,1]
}
p18_results_2       <- sapply(ms,gen_fun_approach_p18)
```


Expected value
1
2
3
4
5
0
20
40
60
80
100
m

19 Electing Emperors and Popes

My attempt

Firstly, the close form solution for a simple form of the puzzle is

$$\binom{N}{1}\left(\sum_{k=M}^{N}\binom{N}{k}(1/N)^k(1-1/N)^{N-k}\right)$$

```
N    <- 7
M    <- 4
k    <- M:N
N*sum(choose(N,k)*((1/N)^(k))*(1 - 1/N)^(N-k))

## [1] 0.07105
```

```
trial_self_voting  <- function(N, M, n){
  candidates       <- 1:N
  biased_list      <- sample(candidates, n)
  mean(replicate(100000, any(table(sample(biased_list, N, replace=TRUE))>=M)))
}
trial_no_self_voting  <- function(N, M, n){
  candidates          <- 1:N
  biased_list         <- sample(candidates, n)
  get_votes           <- function(){
  sapply(1:N, function(z) sample(biased_list[biased_list!=z],1))
 }
 mean(replicate(100000, any(table(get_votes())>=M)))
}
trial_self_voting(N,M,N)

## [1] 0.07053
```

```
puzzle_19_results_1  <- sapply(c(2, 3, 4), function(z)trial_self_voting(7, 4, z))
puzzle_19_results_2  <- sapply(c(2, 3, 4), function(z)trial_self_voting(25, 17, z))
puzzle_19_results_3  <- sapply(c(2, 3, 4), function(z)trial_no_self_voting(25, 17, z))
```

	N = 7	N = 17(self voting)	N = 17(no self voting)
2	1.00000000	0.10797000	0.04877000
3	0.52008000	0.00123000	0.00089000
4	0.28249000	0.00003000	0.00004000

Learning

The author gives additional reference to papers that derive closed form solutions.

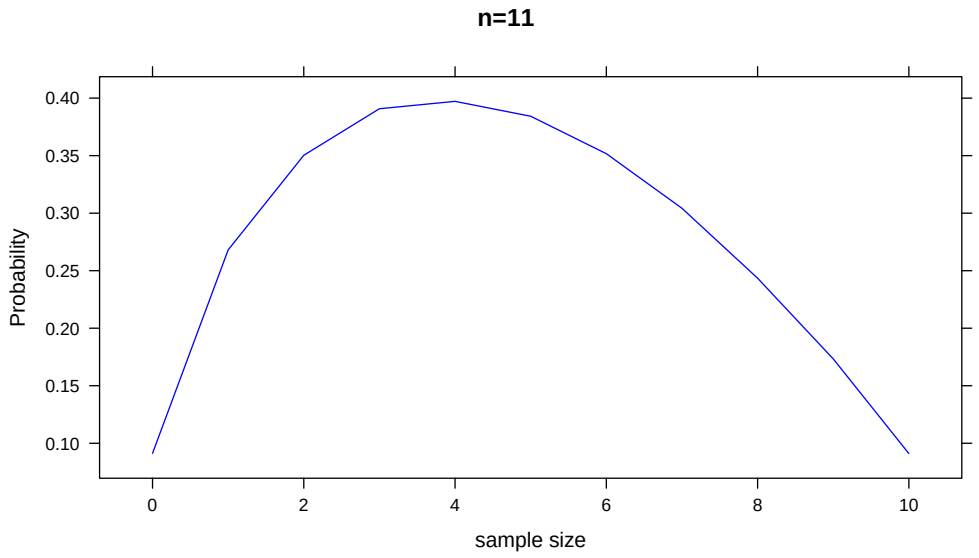
20 An Optimal Stopping Problem

My attempt

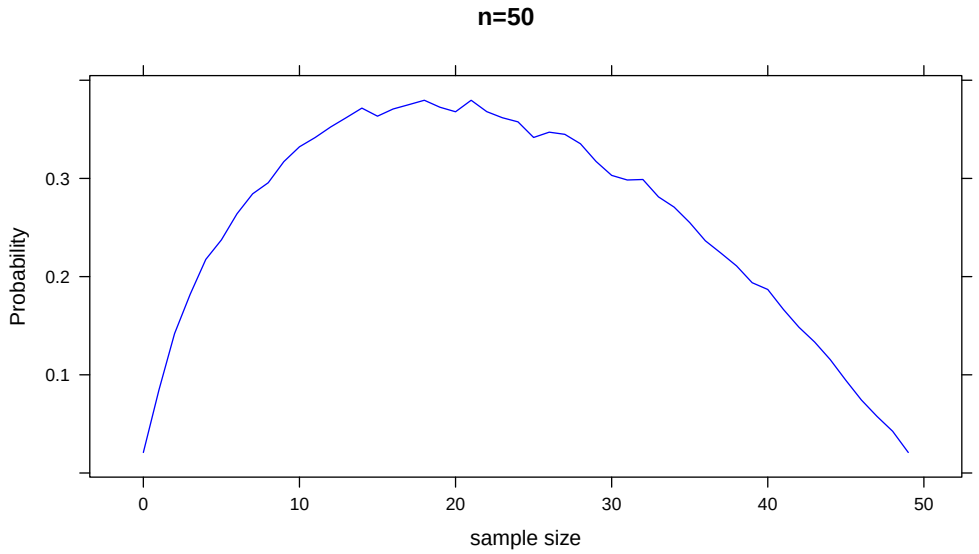
For $n = 11$, what is the probability of choosing the best for various sample sizes

```
best_partner    <- function(m,n){
  draw          <- sample(n)
  if(m==0)  return(draw[1]==1)
  if(m==(n-1)) return(draw[n]==1)
  idx           <- which(draw[(m+1):n] < min(draw[1:m]))[1]
  if(is.na(idx)) return(FALSE)
  draw[(m+1):n][idx]==1
}
n                 <- 11
m11               <- 0:(n-1)
n_11_sim          <- sapply(m11, function(z) mean(replicate(120000,best_partner(z,n))))
n_11_sim          <- data.frame("sample" = m11, "probability" = n_11_sim)

n                 <- 50
m50               <- 0:(n-1)
n_50_sim          <- sapply(m50, function(z) mean(replicate(10000,best_partner(z,n))))
n_50_sim          <- data.frame("sample" = m50, "probability" = n_50_sim)
```

sample	probability
0	0.0913
1	0.2681
2	0.3503
3	0.3908
4	0.3972
5	0.3842
6	0.3517
7	0.3043
8	0.2435
9	0.1733
10	0.0910



Maximum occurs for the sample size : 4



Maximum occurs for the sample size : 18

What happens as $n \to \infty$

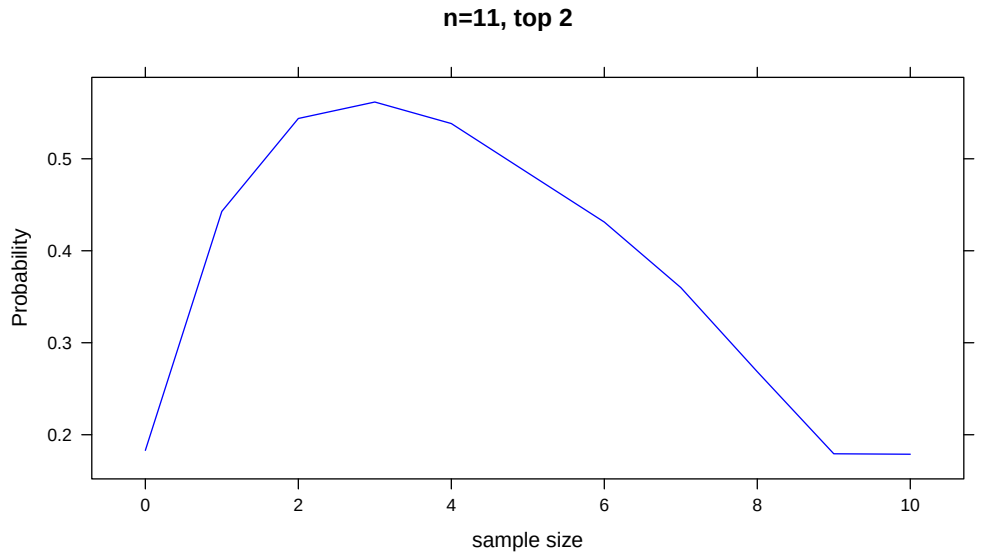
```
get_asymptote     <- function(n){
m_temp             <- 0:(n-1)
n_temp_sim         <- sapply(m_temp, function(z) mean(replicate(10000,best_partner(z,n))))
c(m_temp[which.max(n_temp_sim)],max(n_temp_sim))
}
ns            <- c(5, 10, 20, 50, 100)
asymp_results <- t(sapply(ns, get_asymptote))
asymp_results <- as.data.frame(cbind(ns,asymp_results))
colnames(asymp_results) <- c("n", "sample","prob")
asymp_results$ratio    <- with(asymp_results,n/sample)
```

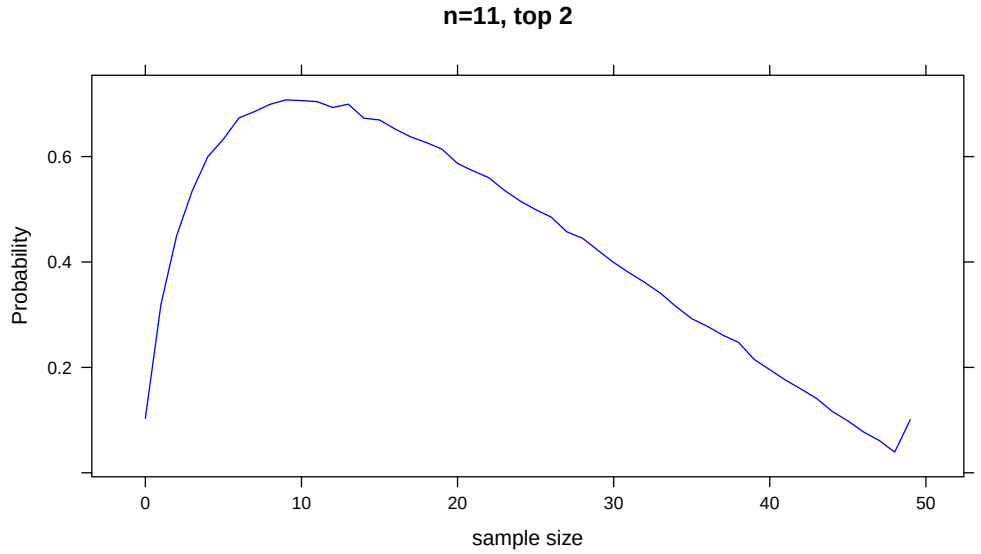
n	sample	prob	ratio
5	2	0.4358	2.5000
10	3	0.4056	3.3333
20	8	0.3827	2.5000
50	18	0.3775	2.7778
100	36	0.3780	2.7778

For $n = 11$, what is the probability of choosing from top $2/3/4/5$ for various sample sizes

```
best_partner_range  <- function(m,n,top){
  draw              <- sample(n)
  if(m==0)     return(draw[1]%in%(1:top))
  if(m==(n-1)) return(draw[n]%in%(1:top))
  idx               <- which(draw[(m+1):n] < min(draw[1:m]))[1]
  if(is.na(idx)) return(FALSE)
  draw[(m+1):n][idx]%in%(1:top)
}

cases1             <- cbind(11,rep(0:10,each=5),rep(1:5,times=11))
cases2             <- cbind(50,rep(0:49,each=5),rep(1:5,times=50))
n_cases1_sim      <- apply(cases1,1,
                           function(z) mean(replicate(10000,
                                                      best_partner_range(z[2],z[1],z[3] )))
                           )
n_cases2_sim      <- apply(cases2,1,
                           function(z) mean(replicate(10000,
                                                      best_partner_range(z[2],z[1],z[3] )))
                           )
puzzle_20_results          <- cbind(cases1, n_cases1_sim)
puzzle_20_results          <- rbind(puzzle_20_results,cbind(cases2, n_cases2_sim))
colnames(puzzle_20_results) <- c("n", "sample", "top", "prob")
puzzle_20_results          <- as.data.frame(puzzle_20_results)
```

```
puzzle_20_best_results      <- ddply(puzzle_20_results,.(n,top),function(z){
                            z[which.max(z$prob),c("sample","prob")]})
```





These figures tell us that the curves of the probability of happiness versus the size of the sample lot have broad maximums, which means that the value of the optimal sample lot size is not much more effective than are its near neighbors. The bar graphs rise monotonically to their maximums and remain near those maximums over an interval of values for the sample lot size, and then monotonically fall.

For each population size and various criterion for evaluation, i.e. top 2/3/4/5, what are the optimal sample size and probabilities ?

n	top	sample	prob
11	1	4	0.3947
11	2	3	0.5617
11	3	2	0.6525
11	4	2	0.7252
11	5	2	0.7736
50	1	18	0.3790
50	2	16	0.5269
50	3	13	0.6159
50	4	10	0.6675
50	5	9	0.7075

Learning

The author computes the closed form probability of getting the best for a sample size r, given the population is n

$$\phi_n(r) = \frac{r-1}{n} \sum_{j=r}^{n} \frac{1}{j-1}$$

For a given n, the above function needs to be maximized to obtain the optimal sample size.

```
obj_fun <- function(r,n){
  (r-1)/n *sum( 1/((r:n)-1))
}
ns            <- c(5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000)
asymp_results <- sapply(ns, function(z){
    res       <- optimize(obj_fun, interval=c(2,z), n=z, maximum=TRUE)
    c(res$maximum,res$objective)
})
asymp_results           <- as.data.frame(cbind(ns,t(asymp_results)))
colnames(asymp_results) <- c("n", "sample","prob")
asymp_results$ratio     <- with(asymp_results,n/sample)
asymp_results           <- asymp_results[,c("n","ratio","prob")]
```

n	ratio	prob
5	1.2500	0.35000
10	2.5000	0.39869
20	2.2222	0.38195
50	2.5000	0.37396
100	2.5641	0.37080
200	2.6667	0.36945
500	2.6316	0.36836
1000	2.7027	0.36819
2000	2.7064	0.36804
5000	2.7115	0.36794

21 Chain Reactions, Branching Processes, and Baby Boys

My attempt

```
set.seed(1)
probs                       <- c(0.4825, 0.2126*(0.5893)^(0:7))
m                           <- 100000
gen                         <- 10
results                     <- matrix(0, nrow = gen, ncol = m)
j                           <- 1
for(j in 1:m){
  current                   <- 1
  results[current , j ]     <- sample(0:8, 1, T, probs)
  survived                  <- sum(results[current , j ])
  while(survived!=0){
     current                <- current + 1
     if(current > gen) break
     results[current , j ] <- sum(sample(0:8, survived, T, probs))
     survived               <- results[current , j ]
  }
}
puzzle_21_results           <- data.frame(
  cases = c(" 2 males in second gen","4 males in second gen", "6 males in third gen"),
  probability = c(mean(results[2, ] == 2),mean(results[2,] == 4), mean(results[2, ] == 6))
      )
```

cases	probability
2 males in second gen	0.0675
4 males in second gen	0.0408
6 males in third gen	0.0233

Learning

Besides the simulation results, the author uses recursive generating function approach to derive the closed form solution for the above puzzle. If one assumes that the generating function at 0^{th} generation is

$$f_1(s) = p_0 + p_1 s + p_2 s^2 + \ldots$$

and define the generating function of n^{th} generation is $f_n(s) = f(f_{n-1}(s))$, the the coefficient of s^k in the n^{th} generation gives the probability that there will be k descendents in the n^{th} generation.