Math 611 HW3

Jeff Gould

9/16/2020

1 Hope College

Attached is a file containing the heights of men and women at Hope College, see file for details. Consider the two component Gaussian mixture model,

$$X = \begin{cases} \mathcal{N}(\mu_1, \sigma_1^2) & \text{with probability } p_1 \\ \mathcal{N}(\mu_2, \sigma_2^2) & \text{with probability } 1 - p_1 \end{cases} \tag{1}$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution and X models the height of a person when gender is unknown. Let $\theta = (\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, p_1)$.

a) The data file specifies gender, but pretend you don't have this information. Write down the log-likelihood function $\ell(\theta)$ and $\nabla\ell(\theta)$ given the height samples, i.e. in terms of $\hat{X}_i$. Write R (or Python) functions that calculate $\ell(\theta)$ and $\nabla\ell(\theta)$

$$L(\theta) = \prod_{i=1}^{N} P(\hat{X}_i|\theta) \Rightarrow \log L(\theta) = \ell(\theta) = \sum_{i=1}^{N} \left[\log \left(p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} + (1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)} \right) \right]$$

$$\frac{\partial \ell}{\partial p_1} = \sum_{i=1}^{N} \left(\frac{\frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} - \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)}}{p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} + (1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)}} \right)$$

$$\frac{\partial \ell}{\partial \mu_1} = \sum_{i=1}^{N} \left(\frac{p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} \cdot \frac{\hat{X}_i - \mu_1}{\sigma_1^2}}{p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} + (1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)}} \right)$$

$$\frac{\partial \ell}{\partial \mu_2} = \sum_{i=1}^{N} \left(\frac{(1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)} \cdot \frac{\hat{X}_i - \mu_2}{\sigma_2^2}}{p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} + (1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)}} \right)$$

$$\frac{\partial \ell}{\partial \sigma_1^2} = \sum_{i=1}^{N} \left(\frac{-(p_1 e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)}) \cdot \frac{1}{2\sigma_1^2\sqrt{2\pi\sigma_1^2}} + p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} \cdot \frac{(\hat{X} - \mu_1)^2}{2(\sigma_1^2)^2}}{p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} + (1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)}} \right)$$

$$\frac{\partial \ell}{\partial \sigma_2^2} = \sum_{i=1}^{N} \left(\frac{-((1-p_1) e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)}) \cdot \frac{1}{2\sigma_2^2\sqrt{2\pi\sigma_2^2}} + (1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)} \cdot \frac{(\hat{X} - \mu_2)^2}{2(\sigma_2^2)^2}}{p_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-(\hat{X}_i - \mu_1)^2/(2\sigma_1^2)} + (1-p_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-(\hat{X}_i - \mu_2)^2/(2\sigma_2^2)}} \right)$$

$$\nabla \ell = \begin{pmatrix} \frac{\partial \ell}{\partial \mu_1} \\ \frac{\partial \ell}{\partial \mu_2} \\ \frac{\partial \ell}{\partial \sigma_1} \\ \frac{\partial \ell}{\partial \sigma_2} \\ \frac{\partial \ell}{\partial p_1} \end{pmatrix}$$

b) Find the MLE for θ by:

- Applying a steepest ascent iteration $\theta^{(i+1)} = \theta^{(i)} + s\nabla\ell(\theta)$.
- Using `nlm` or an equivalent in Python

```
hopeHeights <- read.table("Hope Heights.txt", header = TRUE)

logL <- function(theta, X){

  mu_1 <- theta[1]
  mu_2 <- theta[2]
  sigma_1 <- theta[3]
  sigma_2 <- theta[4]
  p_1 <- max(theta[5], 0) ### constraint in case the step pushed p into negative values
  p_1 <- min(1, p_1) ### constraint in case the step pushes p greater than 1

  logLoss <- sum(
    log(
      p_1 * dnorm(X, mean = mu_1, sd = sqrt(sigma_1)) + (1 - p_1) * dnorm(X, mean = mu_2, sd = sqrt(sig
    )
  )
  return(logLoss)

}


grad_logL <- function(theta, X){

  mu_1 <- theta[1]
  mu_2 <- theta[2]
  sigma_1 <- theta[3]
  sigma_2 <- theta[4]
  p_1 <- theta[5]
  p_2 <- 1 - p_1

  N1 <- dnorm(X, mean = mu_1, sd = sqrt(sigma_1))
  N2 <- dnorm(X, mean = mu_2, sd = sqrt(sigma_2))

  denominator <- p_1 * N1 + p_2 * N2

  d_mu_1 <- (p_1 * N1 *(X - mu_1) / sigma_1) / denominator


  d_mu_2 <- (p_2 * N2 *(X - mu_2) / sigma_2) / denominator
```

```r
  d_sigma_1 <- -(p_1 * (sigma_1 - (X - mu_1)^2)) / (2 * sigma_1^2) * N1 / denominator


  d_sigma_2 <- -(p_2 * (sigma_2 - (X - mu_2)^2)) / (2 * sigma_2^2) * N2 / denominator

  d_p_1 <- (N1 - N2) / denominator


   grad_samples = matrix(c(d_mu_1,
                           d_mu_2,
                           d_sigma_1,
                           d_sigma_2,
                           d_p_1),
                         nrow = length(X),
                         ncol = length(theta))
   gradient <- colSums(grad_samples)
   return(gradient)
}


grad_logL <- function(X, theta)
{
  mu1 <- theta["mu1"]
  mu2 <- theta["mu2"]
  s1 <- theta["s1"]
  s2 <- theta["s2"]
  p1 <- theta["p1"]
  p2 <- 1-p1

  N1 <- dnorm(X, mean=mu1, sd=sqrt(s1))
  N2 <- dnorm(X, mean=mu2, sd=sqrt(s2))
  D <- p1 * N1 + p2 * N2

  grad_samples <- matrix(1/D * c(mu1 = p1 * (X - mu1)/ s1 * N1,
                                 mu2 = p2 * (X - mu2)/ s2 * N2,
                                 s1= p1 * (-1 / s1 + (X - mu1)^2 / (s1^(3/2)))*N1,
                                 s2= p2 * (-1 / s2 + (X - mu2)^2 / (s2^(3/2)))*N2,
                                 p1= N1 - N2),
                               nrow = length(X),
                               ncol = length(theta))

  grad <- grad_samples %>% colSums %>% setNames(c("mu1", "mu2", "s1", "s2", "p1"))
  return (grad_samples %>% colSums)
}


norm <- function(x){
  sqrt(sum(x^2))
}


steepest_ascent <- function(start_theta,
```

```r
                            X = hopeHeights$Height,
                            step = 0.01,
                            epsilon = 1e-3,
                            max_iter = 1e6){

  theta <- start_theta %>% setNames(c("mu1", "mu2", "s1", "s2", "p1"))
  iter <- 0

  current_grad <- grad_logL(theta = theta, X = X)
  loss_history <- c()
  while(norm(current_grad) > epsilon & iter < max_iter){
    iter <- iter + 1

    d <- current_grad / norm(current_grad)
    s <- step
    currentLogL <- logL(theta = theta, X = X)
    loss_history <- c(loss_history, currentLogL)

    while(logL(theta = (theta + s*d), X = X) < currentLogL){
      s <- s/2
    }

    theta <- theta + s * d
    theta[5] <- max(theta[5], 0)
    theta[5] <- min(theta[5], 1)
    current_grad <- grad_logL(theta = theta, X = X)
    if(iter %% 10000 == 0){print(iter)}
  }

  return(list(theta=theta, iter=iter, gradient=current_grad, loss_history = loss_history))

}

set.seed(1)
tictoc::tic()
test <- steepest_ascent(c(runif(2, 60, 85),
                          10,
                          6,
                          0.5), max_iter = 100000)
```

```
## [1] 10000
## [1] 20000
## [1] 30000
## [1] 40000
## [1] 50000
## [1] 60000
## [1] 70000
## [1] 80000
## [1] 90000
## [1] 1e+05
```

```r
tictoc::toc()
```

```
## 104.357 sec elapsed
```

```r
test[["theta"]]
```

```
##        mu1        mu2         s1         s2         p1
## 66.4106684 70.5893162 12.5592296 12.8720695  0.2889779
```

```r
nlm(f = logL, p = 5, X = hopeHeights$Height)
```

```
## Warning in nlm(f = logL, p = 5, X = hopeHeights$Height): NA/Inf replaced by
## maximum positive value

## Warning in nlm(f = logL, p = 5, X = hopeHeights$Height): NA/Inf replaced by
## maximum positive value

## $minimum
## [1] 1.797693e+308
##
## $estimate
## [1] 5
##
## $gradient
## [1] 0
##
## $code
## [1] 1
##
## $iterations
## [1] 0
```

We get a $\theta = (66.41, 70.59, 12.56, 12.87, 0.289)^T$. We did decent in converging to the means, but for some reason have been unable to converge to the correct variances or p_1

$\mu_1 = 66.4$

$\mu_2 = 72.42$

$\sigma_1^2 = 8.5306122$

$\sigma_2^2 = 7.1057143$

$\mu_1 = 0.5$

So we did pretty well with the means, but not great with the σ's or p_1

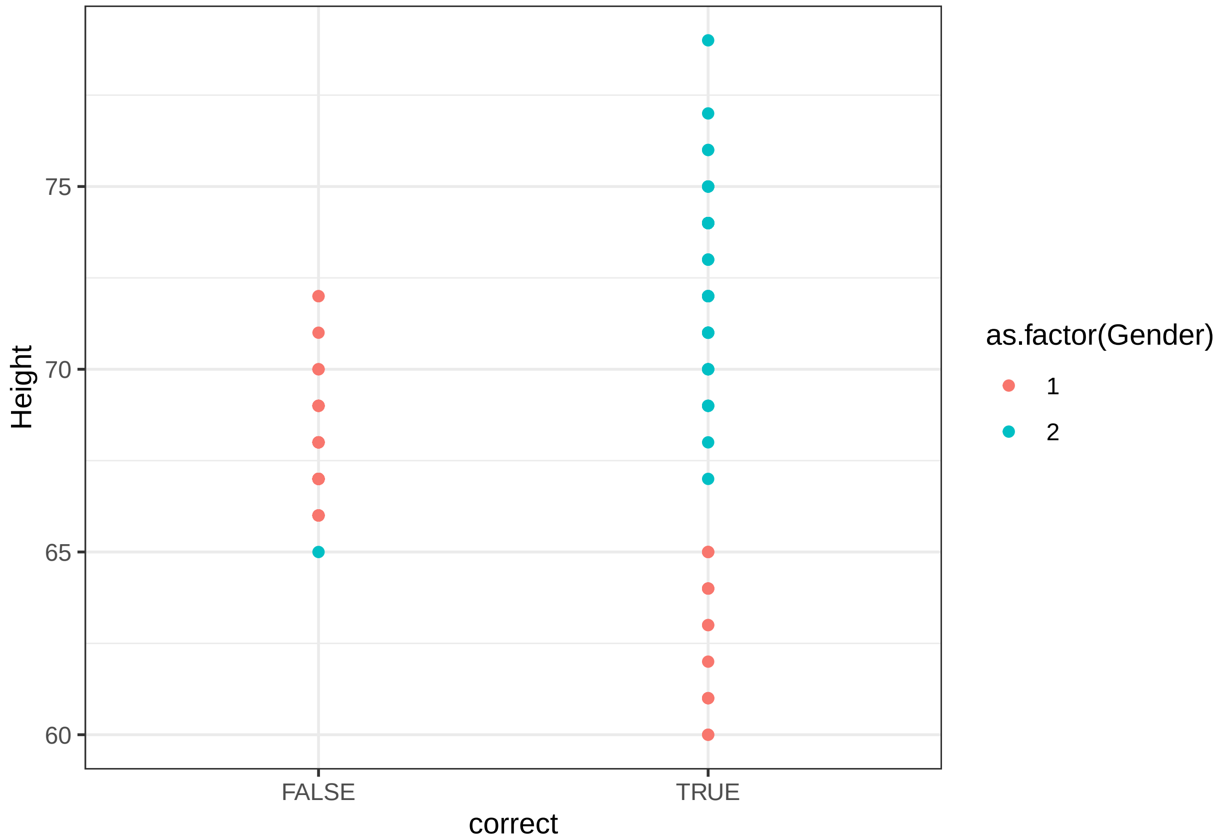
`nlm` does not work with our data set due to the boundary conditions of p

c) Given your MLE in (b), use the distribution of X to predict whether a given sample is taken from a man or woman. Intuitively, the two normal distributions of X correspond to the male and female height distributions. Given a sample, $\hat{X}$, decide which normal the sample is most likely to come from and assign the gender accordingly. Determine what percentage of individuals are classified correctly.

```r
theta <- test[["theta"]]

mu_1 <- theta[1]
mu_2 <- theta[2]
sigma_1 <- theta[3]
sigma_2 <- theta[4]
p_1 <- theta[5]

X_hat <- hopeHeights %>%
  mutate(
    p_female = p_1 * dnorm(Height, mu_1, sqrt(sigma_1)) / (
      p_1 * dnorm(Height, mu_1, sqrt(sigma_1)) +  (1 - p_1) * dnorm(Height, mu_2, sqrt(sigma_2))
    ),
    p_male = (1 - p_1) * dnorm(Height, mu_2, sqrt(sigma_2)) / (
      p_1 * dnorm(Height, mu_1, sqrt(sigma_1)) +  (1 - p_1) * dnorm(Height, mu_2, sqrt(sigma_2))
    )
  ) %>%
  mutate(gender_guess = ifelse(p_female > p_male, 1, 2)) %>%
  mutate(correct = ifelse(gender_guess == Gender, T, F))

mean(X_hat$correct)
```

```
## [1] 0.64
```

```r
ggplot(X_hat, aes(x = correct, y = Height)) +
  geom_point(aes(color = as.factor(Gender))) +
  theme_bw()
```



We correctly classified 0.64 of the samples.

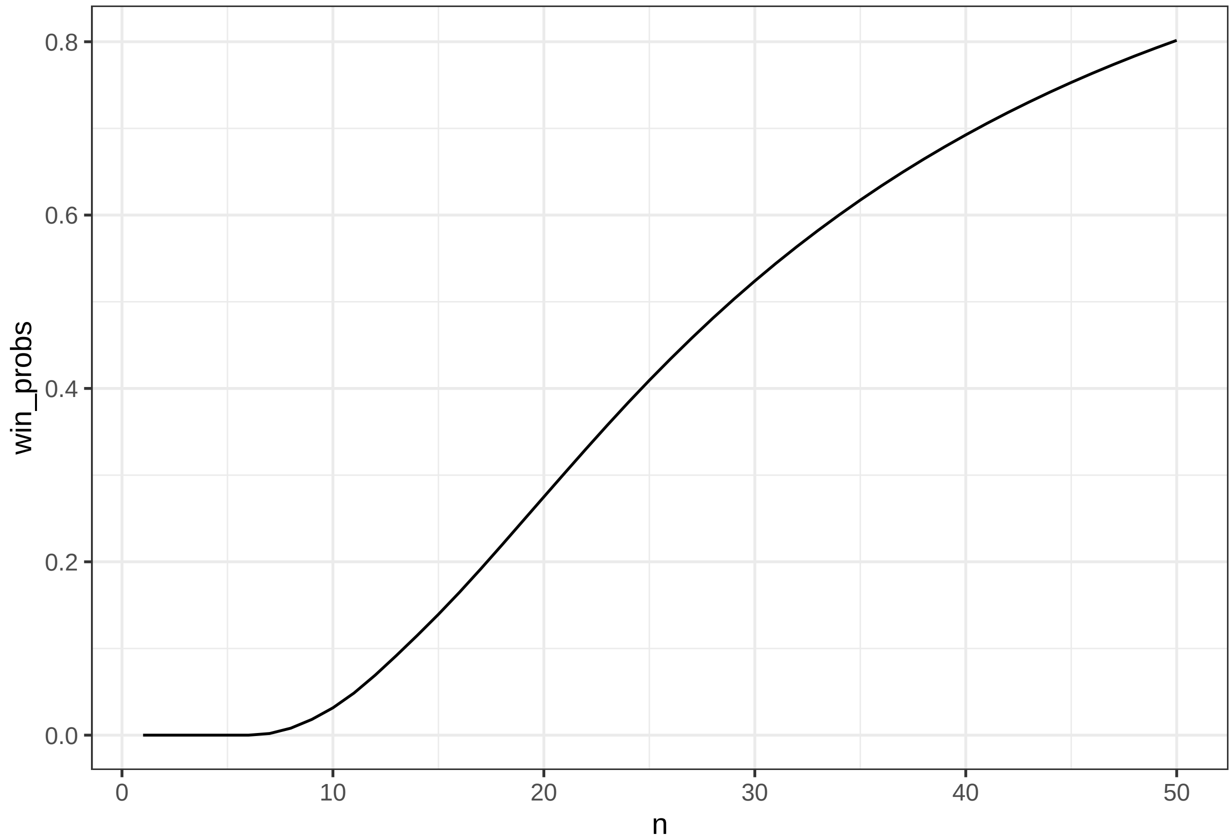
2 Chutes & Ladders (Again)

a) Compute the probabilty of winning in n steps for $n = 1, 2, \ldots, 50$. Compute exactly, do not use Monte Carlo.

The probabilty of ending the game at step n is $P(t_n = 100|t_o = 0)$. For our 101×101 probability matrix (100 spaces and starting off the board) $\underline{P}$, the probabilty at turn n of the game being over is $\underline{P}^n_{1,101}$

```
P <- t(sapply(X = c(0:100), FUN = function(x){
  if(x == 0){
    return(c(0, rep(1/6, 6), rep(0, 94)))
  }else if(x <= 94){
    return(c(rep(0, x+1), rep(1/6, 6), rep(0, 100 - x - 6)))
  }else if(x < 100){
    return(c(rep(0,x+1), rep(1/6, 100-x-1), (6 - (100-x-1)) / 6))
  } else{
    return(c(rep(0,100), 1))
  }
}))

chutes_ladders <- readr::read_csv(file = "~/Desktop/Math-611/HW 2/chutes_and_ladder_locations.csv")


for (i in 1:nrow(chutes_ladders)) {
```

```r
  location <- chutes_ladders$start[i]
  new_end <- chutes_ladders$end[i]

  P_events <- which(P[,location + 1] != 0)
  probs <- P[P_events, location + 1]

  P[P_events, new_end + 1] <- probs + P[P_events, new_end + 1]
  P[P_events, location + 1] <- 0

}

require(expm)

win_probs <- sapply(1:50, function(x){
  P_n = (P %^% x)
  return(P_n[1, 101])
})

Chutes_cdf <- data.frame(
  n = c(1:50),
  win_prob = win_probs
)

ggplot(data = Chutes_cdf, aes(x = n, y = win_probs)) +
  geom_line() +
  theme_bw()
```



b) What's the probabilty of the game lasting more than 1,000 moves?

To solve this, simply take $1 - \underline{P}^{1000}_{1,101}$

```
P_trans <- P %^% 1000

1 - P_trans[1,101]
```

```
## [1] 4.440892e-16
```

c) Suppose we modify Chutes and Ladders so that the player wraps back to square 1 once they go beyond 100. So for example, if the players in on square 99 and rolse a 3, they end up on square 2.

```
P <- t(sapply(X = c(0:100), FUN = function(x){
  if(x == 0){
    return(c(0, rep(1/6, 6), rep(0, 94)))
  }else if(x <= 94){
    return(c(rep(0, x+1), rep(1/6, 6), rep(0, 100 - x - 6)))
  }else if(x <= 100){
    return(c(0, rep(1/6, (x + 6)%%100), rep(0, 94), rep(1/6, 100-x)))
  } else{
    return(c(rep(0,100), 1))
  }
```

```r
}))

for (i in 1:nrow(chutes_ladders)) {
  location <- chutes_ladders$start[i]
  new_end <- chutes_ladders$end[i]

  P_events <- which(P[,location + 1] != 0)
  probs <- P[P_events, location + 1]

  P[P_events, new_end + 1] <- probs + P[P_events, new_end + 1]
  P[P_events, location + 1] <- 0

}
```

i) Compute the stationary distribution. After 1 billion moves, on what square is a player's piece most likely to be on?

The stationary distribution π for a transtion probabilty matrix $\underline{P}$ is simply a row vector of $\lim_{n\to\infty} \underline{P}^n$

```r
stationaryP <- P %^% 1e9

Pi <- stationaryP[1,2:101]

which.max(Pi)
```

```
## [1] 44
```

```r
Pi[which.max(Pi)]
```

```
## [1] 0.03039505
```

We see that after 1 billion turns, the player is most likely to be on square 44, with $P = 0.0303$

ii) Compute the relaxation time and explain why it makes sense in terms of your results for the expected time of a game in the previous HW.

relaxation time $= \frac{1}{1-|\lambda_2|}$

```r
eigP <- eigen(P)
#head(Re(eigP$values))

1 / (1 - abs(Re(eigP$values[2])))
```

```
## [1] 4.991908
```

The relaxation time is indicative of how long it takes to converge to the stationary distribution, so every 5.59 turns the time remaining to converge decreases by a factor of $1/e \approx 0.368$. Considering the average game lasts about 35 turns, from HW2, this seems faster than we may expect. This suggests that perhaps the randomness of the chutes and ladders quickly move a player around the board, creating faster convergence.

iii) Now suppose that we play the game as follows. Each round, with probaility p, the player doesn't move. With probability $1 - p$ the player rolls the die and moves as usual. For $p = .9, .99, .999$, compute the relaxation times

```
modifiedP <- function(p){
  P <- t(sapply(X = c(0:100), FUN = function(x){
  if(x == 0){
    return(c(0, rep((1-p)/6, 6), rep(0, 94)))
  }else if(x <= 94){
    return(c(rep(0, x+1), rep((1-p)/6, 6), rep(0, 100 - x - 6)))
  }else if(x <= 100){
    return(c(0, rep((1-p)/6, (x + 6)%%100), rep(0, 94), rep((1-p)/6, 100-x)))
  } else{
    return(c(rep(0,100), 1))
  }
}))

  for (i in 1:nrow(chutes_ladders)) {
    location <- chutes_ladders$start[i]
    new_end <- chutes_ladders$end[i]

    P_events <- which(P[,location + 1] != 0)
    probs <- P[P_events, location + 1]

    P[P_events, new_end + 1] <- probs + P[P_events, new_end + 1]
    P[P_events, location + 1] <- 0

  }
  P <- P + diag(p, nrow = 101, ncol = 101)
  return(P)
}

P.9 <- modifiedP(0.9)
P.99 <- modifiedP(0.99)
P.999 <- modifiedP(0.999)

relaxationTime <- function(X){
  eigX <- eigen(X)
  return(1 / (1 - abs(Re(eigX$values[2]))))
}

relaxationTime(P.9)
```

```
## [1] 49.91908
```

```
relaxationTime(P.99)
```

```
## [1] 499.1908
```

```
relaxationTime(P.999)
```

```
## [1] 4991.908
```

As p increases, and we have less movement with each turn, the relaxation time increases, approximately by a factor of 10 with each example given. This indicates that it takes longer to converge to the stationary distribution, which makes sense as there are a lot more turns without any movement.