Final\_Homwork

##1번 문제

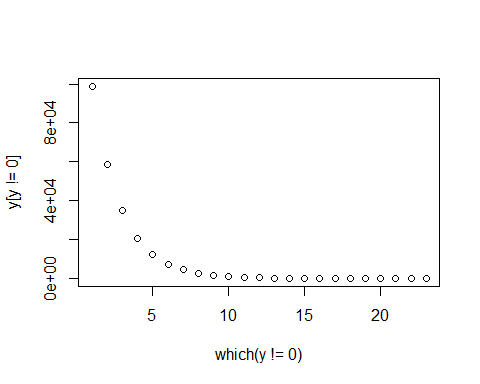
Streaky hitting in baseball (11.5절)의 내용을 수정하여 한게임에 출전하여 1번 이상 안타를 칠 확률이 h인 야구선수가 k게임이 출장하였을 때 **[1] hitting streak의 분포**와 **[2] 최장 hitting streak이 s 보다 클 확률** 을 시뮬레이션을 이용하여 추정하는 프로그램을 작성하고 결과(여러 가지의 h, k, s 값에 대한)와 함께 제출하시오.

**[1]**

streak = function(n, k, h){  
 result = rep(0, k)  
 for (i in 1:n){  
 y = rbinom(k, 1, h)  
 where = c(0, y, 0) == 0  
 n = length(y)  
 loc.zeros = (0:(n + 1))[where]  
 streak.lengths = diff(loc.zeros) - 1  
 streak.lengths = streak.lengths[streak.lengths > 0]  
 for(t in streak.lengths){  
 result[t] = result[t]+1}  
 }  
 result  
}

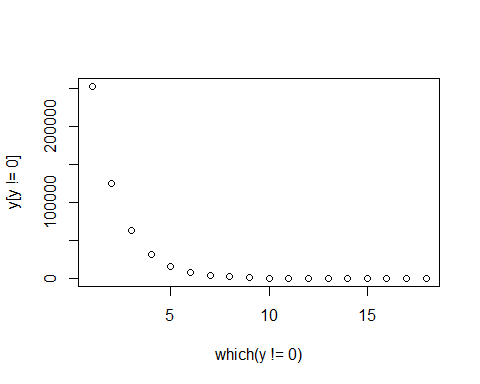
hitting streak의 분포를 시뮬레이션을 통해 구하는 코드이다. replicate 함수를 사용할 경우, 시뮬레이션의 결과값이 list로 저장된다. 이를 적당히 처리하는 방법을 잘 모르기 때문에, for문을 사용하여 시뮬레이션하였다.

y = streak(10000, 100, 0.6)  
plot(which(y != 0), y[y != 0])



k = 100, h = 0.6으로 만번 시뮬레이션 했을 때의 분포는 이렇다. 안타를 10번 연속으로 칠 확률이 그리 높지 않음을 알 수 있다.

y = streak(10000, 200, 0.5)  
plot(which(y != 0), y[y != 0])



k = 200, h= 0.5 일때의 분포이다.

**[2]**

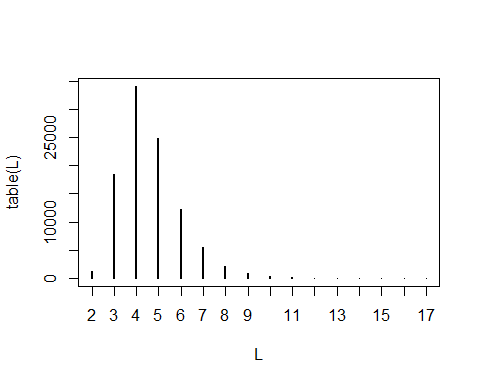
longest.streak <- function(x) {  
 where = c(0, x, 0) == 0  
 n = length(x)  
 loc.zeros = (0 : (n + 1))[where]  
 streak.lengths = diff(loc.zeros) - 1  
 streak.lengths = streak.lengths[streak.lengths > 0]  
 max(streak.lengths)  
}  
random.streak=function(k, h){  
 y = rbinom(k, 1, h)  
 mixed.up.y = sample(y)  
 longest.streak(mixed.up.y)  
}

원래 코드를 일부 수정하였다. random.streak 함수를 만들때, 변수 k와 h를 사용하도록 변경했다. k게임에서, 한 경기에 한 번 이상 안타를 칠 확률 h를 결합하기 위해서다. rbinom(k, 1, h)를 통해 난수추출한 값들을 y에 넣고, 이를

L <- replicate(100000, random.streak(k, h))  
plot(table(L))

이를 10만번 시뮬레이션 하고, 분포를 시각화하기 위해 이런 코드를 사용했다.

L <- replicate(100000, random.streak(k = 100, h = 0.4))  
plot(table(L))



k = 100, h = 0.4일때, 최장 hitting streak는 7을 넘어가기 힘들다.

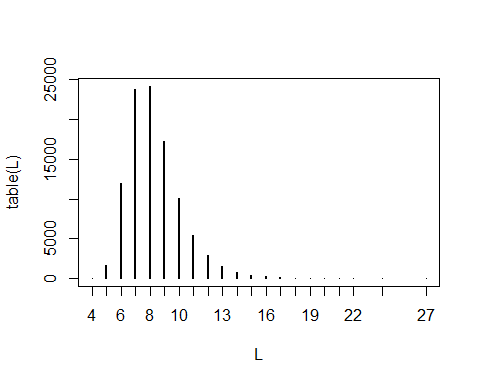
**[2]**

prob.bigger.s <- function(k, h, s) {  
 replicate(10000, random.streak(k, h) > s)  
}

fors <- prob.bigger.s(k, h, s)

최장 hitting streak가 s보다 클 확률을 계산하기 위한 코드다. 기존 random.streak함수에 replicate함수를 이용해 만들었다. prob.bigger.s의 결과값을 fors에 저장한다.

L <- replicate(100000, random.streak(500, 0.5)) # k = 500, h = 0.5  
plot(table(L))



k = 500이고 h= 0.5일때, 이런식의 분포가 생성된다.

fors <- prob.bigger.s(500, 0.5, 10) # s = 10  
table(fors)

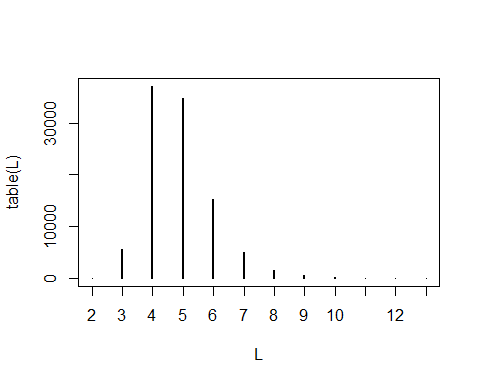
## fors  
## FALSE TRUE   
## 8879 1121

mean(fors)

## [1] 0.1121

최장 안타기록이 10을 넘어갈 확률은 11%정도이다.

L <- replicate(100000, random.streak(500, 0.3)) # k = 500, h = 0.3  
plot(table(L))



k = 500이고 h = 0.3일 때의 분포는 이렇다.

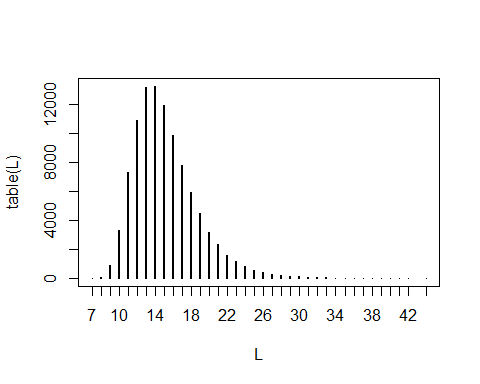
fors <- prob.bigger.s(500, 0.3, 5) # s = 5  
table(fors)

## fors  
## FALSE TRUE   
## 7754 2246

mean(fors)

## [1] 0.2246

L <- replicate(100000, random.streak(500, 0.7)) # k = 500, h = 0.7  
plot(table(L))



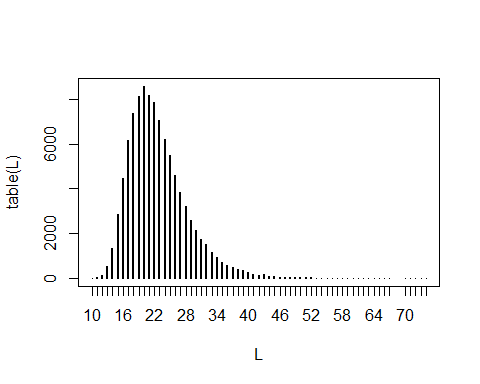
fors <- prob.bigger.s(500, 0.7, 20) # s = 20  
table(fors)

## fors  
## FALSE TRUE   
## 9241 759

mean(fors)

## [1] 0.0759

L <- replicate(100000, random.streak(500, 0.8)) # k = 500, h = 0.8  
plot(table(L))



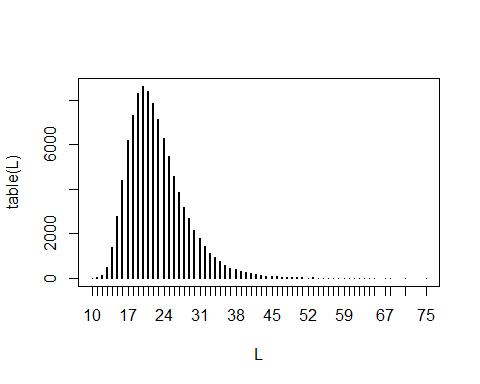
fors <- prob.bigger.s(500, 0.8, 25) # s = 25  
table(fors)

## fors  
## FALSE TRUE   
## 7512 2488

mean(fors)

## [1] 0.2488

L <- replicate(100000, random.streak(500, 0.8)) # k = 500, h = 0.8  
plot(table(L))



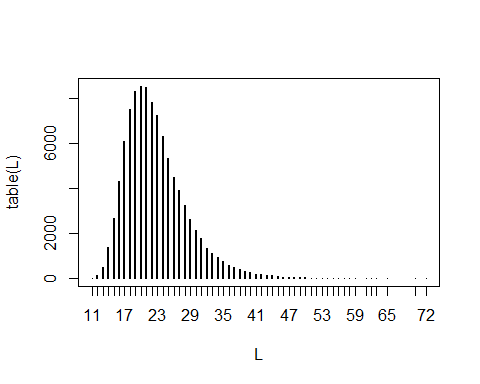
fors <- prob.bigger.s(500, 0.8, 30) # s = 30  
table(fors)

## fors  
## FALSE TRUE   
## 9057 943

mean(fors)

## [1] 0.0943

L <- replicate(100000, random.streak(500, 0.8)) # k = 500, h = 0.8  
plot(table(L))



fors <- prob.bigger.s(500, 0.8, 38) # s = 38  
table(fors)

## fors  
## FALSE TRUE   
## 9849 151

mean(fors)

## [1] 0.0151

## 2번 문제

13.1 (Late to class?). Suppose the travel times for a particular student from home to school are normally distributed with mean 20 minutes and standard deviation 4 minutes. Each day during a five-day school week she leaves home 30 minutes before class. For each of the following problems, write a short Monte Carlo simulation function to compute the probability or expectation of interest.

####**a**

Find the expected total traveling time of the student to school for a fiveday week. Find the simulation estimate and give the standard error for the simulation estimate.

expected <- function(n) {  
 travel.time = rnorm(n, 20, 4)  
 sum.time <- sum(travel.time)  
 sum.time   
}  
expected(5)

## [1] 111.9233

5일 동안의 총 여행시간을 구하기위한 코드다.

simul.expected.all <- replicate(10000, expected(5))  
head(simul.expected.all)

## [1] 111.45433 105.27188 93.77182 111.54812 85.41893 107.34089

simul.expected.total.mean <- mean(simul.expected.all)  
simul.expected.total.mean

## [1] 99.91587

시뮬레이션 결과, 5일동안의 총 여행시간의 기댓값에 대한 시뮬레이션을 통한 추정치는 약 100정도이다. 평균이 20인 정규분포이므로, 어느정도 예측가능한 수준안에서 만들어졌다고 직관적으로 판단할 수 있다.

total.sehat <- sqrt(sum((simul.expected.all - simul.expected.total.mean)^2) / 10000)  
total.sehat

## [1] 9.017616

var은 편차 제곱합을 (n - 1)로 나눈다. 하지만 이는 시뮬레이션을 통한 결과 이기 때문에 n으로 나눠야 하므로, 직접 표준오차를 계산하였다.

계산한 결과, 약 9가 나왔다.

####**b**

Find the probability that the student is late for at least one class in the five-day week. Find the simulation estimate of the probability and the corresponding standard error.

prob.thirty <- function(n) {  
 travel.time = rnorm(n, 20, 4)  
 prob <- sum(travel.time > 30) / n  
 prob  
}  
prob.thirty(5)

## [1] 0

5일동안 늦는 경우에 대한 코드다. 이를 시뮬레이션해보자

simul.late.cases <- replicate(100000, prob.thirty(5))  
simul.late.prob.mean <- mean(simul.late.cases)  
simul.late.prob.mean

## [1] 0.006176

단 한번이라도 늦을 확률은 0.006으로 되게 낮다. 실제로 원래 분포에서 30분 이상으로 도착할 확률은 표준정규분포에서 5/2보다 클 확률이다. 매우 작은 확률이므로, 이 시뮬레이션은 어느정도 믿을만 한 것 같다

late.sehat <- sqrt(sum((simul.late.cases - simul.late.prob.mean)^2) / 100000)  
late.sehat

## [1] 0.03503508

그리고 이때의 표준오차를 구하면, 약 0.035가 나온다.

####**c** On average, what will be the longest travel time to school during the fiveday week? Again find the simulation estimate and the standard error.

longest.time <- function(n) {  
 travel.time = rnorm(n, 20, 4)  
 max(travel.time)  
}

5일동안 가장 긴 travel time을 추출하기위한 코드다. 이를 시뮬레이션 해보면,

simul.longest.cases <- replicate(100000, longest.time(5))  
head(simul.longest.cases)

## [1] 23.81937 22.31846 23.59768 23.05708 28.95021 24.51886

simul.longest.mean <- mean(simul.longest.cases)  
simul.longest.mean

## [1] 24.64485

약 24.5정도가 나옴을 알 수 있고, 정규분포의 형태를 고려할 때, 합리적인 결과로 보인다.

longests.sehat <- sqrt(sum((simul.longest.cases - simul.longest.mean)^2) / 100000)  
longests.sehat

## [1] 2.679249

그리고 이때의 표준오차는 2.67정도가 나온다.

## 3번 문제

13.2 (Confidence interval for a normal mean based on sample quantiles). Suppose one obtains a normally distributed sample of size n=20 but only records values of the sample median M and the first and third quartiles Q1 and Q3.

####**a** Using a sample of size n=20 from the standard normal distribution, simulate the sampling distribution of the statistic S = M / (Q3 - Q1) Store the simulated values of S in a vector.

dis.stat <- function(n) {  
 x <- rnorm(n, 0, 1)  
 Q1 <- quantile(x, 0.25, names = F)  
 Q3 <- quantile(x, 0.75, names = F)  
 sam.med <- median(x)  
 S <- sam.med / (Q3 - Q1)  
 S  
}

표준정규분포의 사분위수들로 만들어진 통계량을 구하는 코드이다. Q1과 Q3의 경우 names = F를 넣어줌으로써, 불필요한 이름을 삭제했다.

dis.stat(n = 20)

## [1] -0.4629556

simul.S <- replicate(10000, dis.stat(n = 20))  
head(simul.S)

## [1] 0.02044043 -0.24595189 -0.05760789 0.31644451 -0.28026485 0.01963177

sample size 20에서 표준정규분포를 뽑았고, 이를 시뮬레이션 한 결과를 simul.S에 저장하였다.

####**b** Find two values, s1,s2, that bracket the middle 90% probability of the distribution of S.

quantile(simul.S, c(0.05, 0.95))

## 5% 95%   
## -0.3854168 0.3778900

시뮬레이션 결과에 대해서 가운데 90%구간의 범위를 quantile함수를 통해 나타냈다.

####**c** For a sample of size n = 20 from a normal distribution with mean μ and standard deviation σ, it can be shown that P(s1 < (M - mu) / (Q3 - Q1) < s2) = 0.90 Using this result, construct a 90% confidence interval for the mean μ

s1 <- -0.3719357  
s2 <- 0.3962538

이전에 구한 quantile 값을 저장하고,

CI.for90 <- c(-s2 \* (Q3 - Q1) + sam.med, -s1 \* (Q3 - Q1) + sam.med)

이전 결과를 이용해, S - 0.95quantile과 S - 0.05quantile을 계산하여 구하는 식이다. Q1, Median, Q3를 알기만 한다면, 이 식에 대입해 구할 수 있다.

####**d** In a sample of 20, we observe (Q1,M,Q3) = (37.8,51.3,58.2). Using your work in parts (b) and (c), find a 90% confidence interval for the mean μ.

s1 <- -0.3719357  
s2 <- 0.3962538  
Q1 <- 37.8  
sam.med <- 51.3  
Q3 <- 58.2

CI.for90.b <- c(-s2 \* (Q3 - Q1) + sam.med, -s1 \* (Q3 - Q1) + sam.med)  
CI.for90.b

## [1] 43.21642 58.88749

주어진 값들을 통해 모평균에 대한 90%의 구간추정을 하면, (43.216, 58.887)가 나온다.

## 4번 문제

13.3 (Comparing variance estimators). Suppose one is taking a sample y1, …,yn from a normal distribution with mean μ and variance σ2.

####**a** It is well known that the sample variance S = sigma (yi - ybar)^2 / (n - 1) is an unbiased estimator of σ2. To confirm this, assume n=5 and perform a simulation experiment to compute the bias of the sample variance S.

samp.var.bias <- function(n, mu, s) {  
 y <- rnorm(n, mu, s)  
 S <- sum((y - mean(y))^2) / (n - 1)  
 s^2 - S  
}

bias를 계산하는 코드를 작성하였다. 이를 가지고 시뮬레이션을 해보면,

simul.bias <- replicate(100000, samp.var.bias(5, 0, 5))  
mean(simul.bias)

## [1] -0.04860833

오차는 0.05정도로 작다.

simul.bias <- replicate(1000000, samp.var.bias(5, 0, 5))  
mean(simul.bias)

## [1] -0.005814938

더 많이 시뮬레이션을 해볼경우, 오차는 0.008로 더 줄어들고, 불편추정량이라고 판단할 수 있다.

simul.bias <- replicate(100000, samp.var.bias(5, 10, 10))  
mean(simul.bias)

## [1] 0.06470519

모분산을 키워서 분포가 더 퍼짐에 따라, 편의도 줄어듬을 볼 수 있고, 불편추정량이라고 말할 수 있다.

####**b** Consider the alternative variance estimator Sc = sigma (yi - ybar)^2 / c where c is a constant. Suppose one is interested in finding the estimator Sc that makes the mean squared error MSE = E[(Sc - σ2)^2] as small as possible.

Again assume n=5 and use a simulation experiment to compute the mean squared error of the estimators *S3,S5,S7,S9* and find the choice of c (among {3, 5, 7, 9}) that minimizes the MSE.

samp.var.mse <- function(n, mu, s, c) {  
 y <- rnorm(n, mu, s)  
 Sc <- sum((y - mean(y))^2) / c  
 mean((Sc - s^2)^2)  
}

S3 <- mean(replicate(10000, samp.var.mse(5, 0, 5, 3)))  
S5 <- mean(replicate(10000, samp.var.mse(5, 0, 5, 5)))  
S7 <- mean(replicate(10000, samp.var.mse(5, 0, 5, 7)))  
S9 <- mean(replicate(10000, samp.var.mse(5, 0, 5, 9)))  
S.vec <- c(S3 = S3, S5 = S5, S7 = S7, S9 = S9)  
S.vec

## S3 S5 S7 S9   
## 608.7056 217.5272 218.9232 252.7348

S.vec[which.min(S.vec)]

## S5   
## 217.5272

비록 S7은 편의추정량이지만, 편의를 발생시킴으로써 줄어드는 분산이 더 크다. 따라서 S7이 가장 MSE를 작게 하는 추정량이다.

## 5번 문제

13.5 (Metropolis-Hastings algorithm for the poly-Cauchy distribution). Suppose that a random variable y is distributed according to the poly-Cauchy density g(y) = prod [1 / {π \* (1+(y−ai)^2)}] where a=(a1, …,an) is a vector of real-valued parameters. Suppose that n=6 and a = (1,2,2,6,7,8).

####**a**

Write a function to compute the log density of y. (It may be helpful to use the function dcauchy that computes the Cauchy density.)

log.g <- function(y) {  
 a <- c(1, 2, 2, 6, 7, 8)  
 log(prod(dcauchy(y, a, 1)))  
}

poly cauchy 분포에 대한 pdf이다

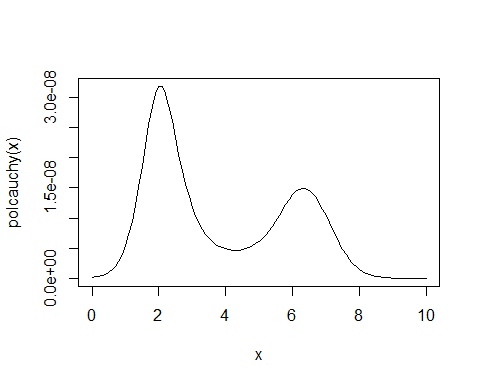
####**b**

Use the function metrop.hasting.rw to take a simulated sample of size 10,000 from the density of y. Experiment with different choices of the standard deviation C. Investigate the effect of the choice of C on the acceptance rate, and the mixing of the chain over the probability density.

metrop.hasting.rw = function(logf, current, C, iter, ...){  
 S = rep(0, iter); n.accept = 0  
 for(j in 1:iter){  
 candidate = rnorm(1, mean=current, sd=C)  
 prob = exp(logf(candidate, ...) - logf(current, ...))  
 accept = ifelse(runif(1) < prob, "yes", "no")  
 current = ifelse(accept == "yes", candidate, current)  
 S[j] = current; n.accept = n.accept + (accept == "yes")  
 }  
 list(S=S, accept.rate=n.accept / iter)  
}

metropolis-hasting 알고리즘에 대한 코드를 변형했다. logf 부분에 우리의 함수를 넣으면 된다. 적절한 acceptance rate는 0.25~0.5 사이이므로, 그 사이 값을 만들어내는 C를 고른다.

polcauchy = function(y){  
 a=c(1,2,2,6,7,8)  
 result = NULL  
 for(x in y){  
 dc = prod(dcauchy(x, location = a))  
 result = c(result, dc)}  
 result  
}  
curve(polcauchy(x),from=0,to=10)

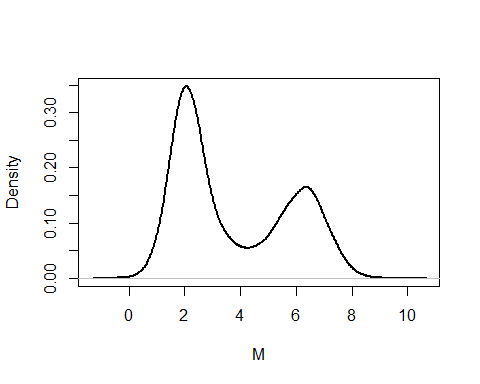


폴리 코시 분포는 대략 이러하고, 이런 분포와 얼마나 유사한지를 살펴보자.

mcmc.sample1 = metrop.hasting.rw(log.g, 1, 1, 10000)  
mcmc.sample1$accept.rate

## [1] 0.6711

plot(density(mcmc.sample1$S), lwd=2, main="", xlab="M")

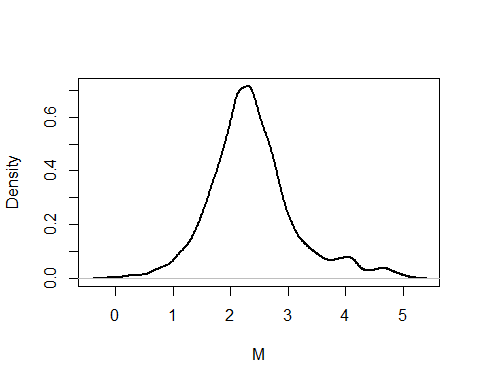


67%로, 우리 기준보다 더 높다.

mcmc.sample2 = metrop.hasting.rw(log.g, 1, 0.1, 10000)  
mcmc.sample2$accept.rate

## [1] 0.9587

plot(density(mcmc.sample2$S), lwd=2, main="", xlab="M")

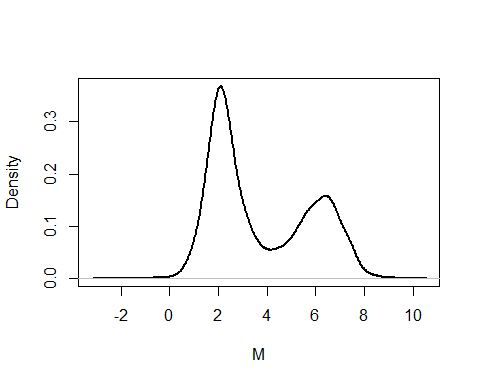


96%로 매우 높아서, C = 0.1은 선택될 수 없다.

mcmc.sample3 = metrop.hasting.rw(log.g, 1, 3, 10000)  
mcmc.sample3$accept.rate

## [1] 0.4366

plot(density(mcmc.sample3$S), lwd=2, main="", xlab="M")

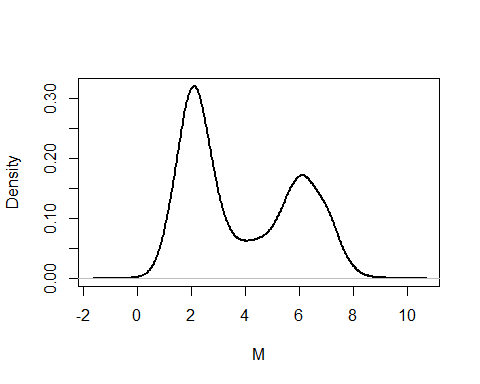


45%로 적절하지만, 조금 더 낮을 값을 선택하고 싶다.

mcmc.sample4 = metrop.hasting.rw(log.g, 1, 5, 10000)  
mcmc.sample4$accept.rate

## [1] 0.3304

plot(density(mcmc.sample4$S), lwd=2, main="", xlab="M")

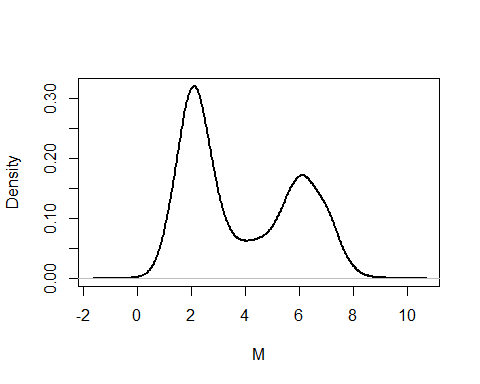


C = 5 일때, acceptance rate는 32%로 적절하며, 그래프도 어느정도 안정적이다.

####**c**

Using the simulated sample from a “good” choice of C, approximate the probability P(6<Y <8).

plot(density(mcmc.sample4$S), lwd = 2, main = "", xlab = "M")



그래프 상으로 P(6 < Y < 8)은 작지않은 확률을 지닐 것 같다.

mean(mcmc.sample4$S > 6 & mcmc.sample4$S < 8)

## [1] 0.2238

이를 시뮬레이션을 통해 계산하면, 0.22정도의 확률이 계산된다.

## 6번 문제

13.6 (Gibbs sampling for a Poisson/gamma model). Suppose the vector of random variables (X,Y ) has the joint density function and we wish to simulate from this joint density.

**a** Show that the conditional density f(x|y) has a gamma density and identify the shape and rate parameters of this density. **b** show that the conditional density f(y|x) has a Poisson density.

*a와 b는 수식으로 증명했다*

*텍스트이(가) 표시된 사진

자동 생성된 설명*

####**c**

Write a R function to implement Gibbs sampling when the constants are given by a = 1 and b = 1.

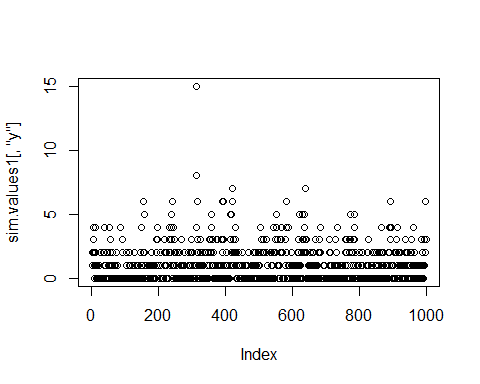
random.coin.gibbs = function(y, m){  
 S = matrix(0, m, 2)  
 dimnames(S)[[2]] = c("y", "x")  
 for(j in 1:m){  
 x = rgamma(1, y + 1, 2)  
 y = rpois(1, x)  
 S[j, ] = c(y, x)  
 }  
 return(S)  
}

X|Y ~ gamma(a + y, 1/(b + 1)) Y|X ~ pois(x) 로 부터 깁스 샘플링을 이용하는 코드다. x = rgamma(1, y + 1, 2) 여기서 1/2대신 2인 것을 생각해두자.

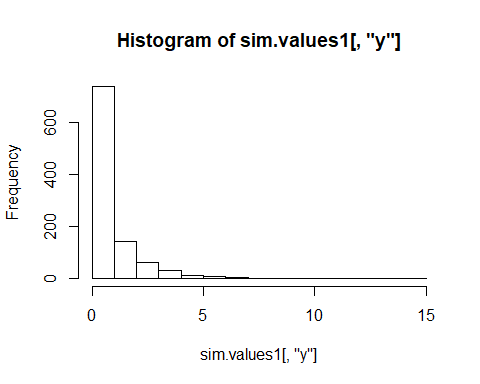
####**d**

Using your R function, run 1000 cycles of the Gibbs sampler and from the output, display (say, by a histogram) the marginal probability mass function of Y and compute E(Y ).

sim.values1 = random.coin.gibbs(y = 1, m = 1000)  
plot(sim.values1[ ,"y"])



hist(sim.values1[ ,"y"])



초기값을 y = 1을 두고 계산했을때의 plot과 히스토그램을 그렸다. 왼쪽에 값이 몰려있다.

table(sim.values1[ ,"y"])

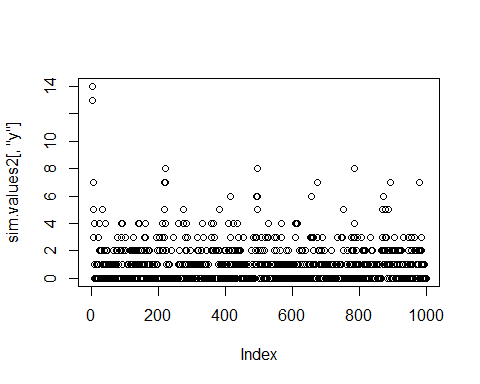
##   
## 0 1 2 3 4 5 6 7 8 15   
## 496 245 141 61 32 12 9 2 1 1

mean(sim.values1[ , "y"])

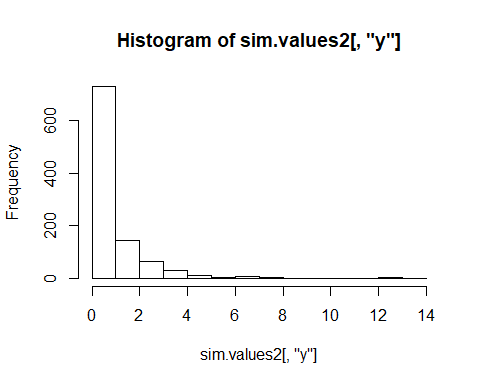
## [1] 0.989

시뮬레이션 결과, 기댓값은 0.95정도이다.

sim.values2 = random.coin.gibbs(y = 30, m = 1000)  
plot(sim.values2[ ,"y"])



hist(sim.values2[ ,"y"])



초기값을 y = 30으로 설정했음에도, 분포에 큰 차이가 나타나지 않는다. 이는 깁스 샘플링의 장점이다.

table(sim.values2[ ,"y"])

##   
## 0 1 2 3 4 5 6 7 8 13 14   
## 474 257 146 65 30 11 5 6 3 2 1

mean(sim.values2[ , "y"])

## [1] 1.055

기댓값은 1정도가 나오고, 이전 시뮬레이션 결과값과 유사하다.