STAT 675 HW #4

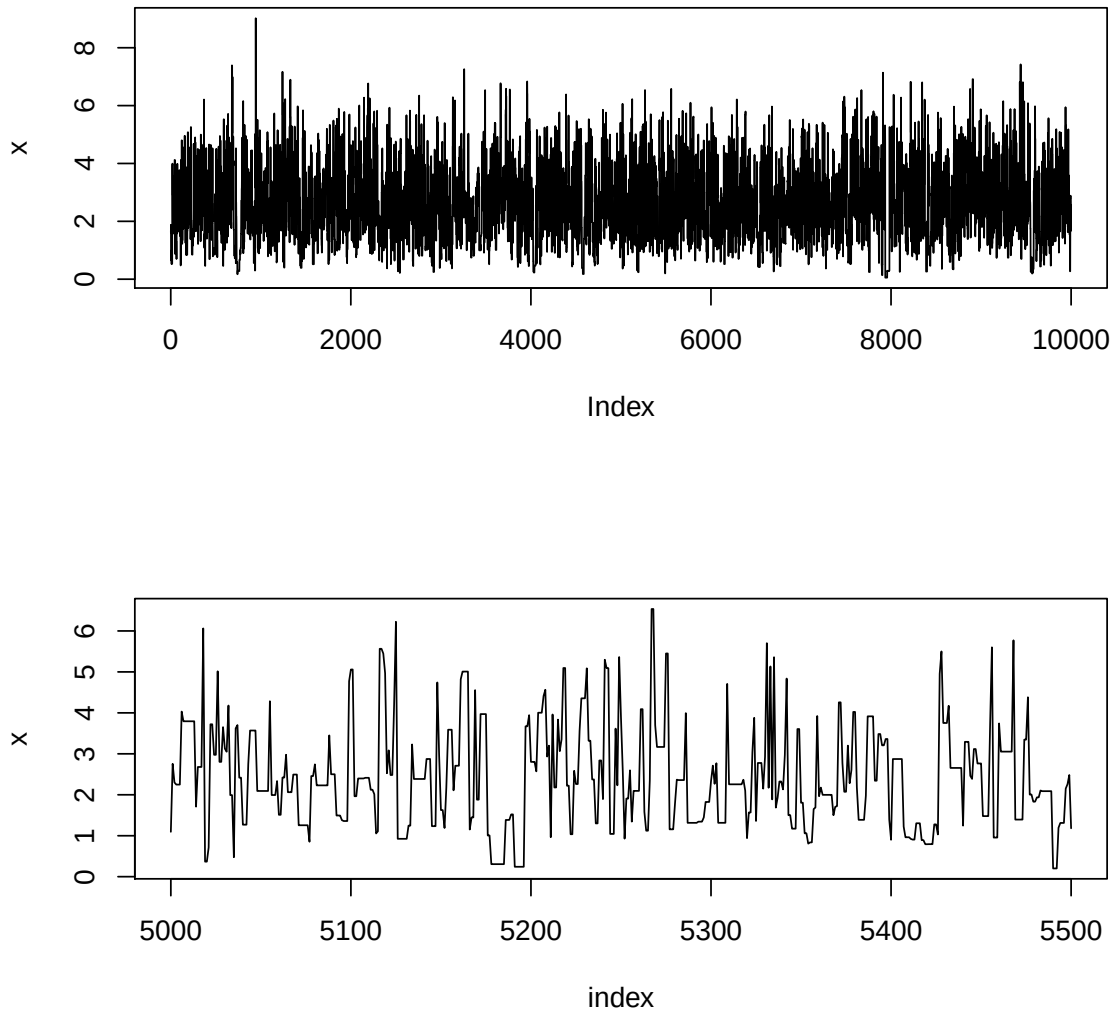
Dominic D LaRoche

April 6, 2014

9.1 Repeat example 9.1.

```
set.seed(36)
f <- function(x, sigma) {
    if (any(x < 0))
        return(0)
    stopifnot(sigma > 0)
    return((x/sigma^2) * exp(-x^2/(2 * sigma^2)))
}

m <- 10000
sigma <- 2  #changed to sigma=2
x <- numeric(m)
x[1] <- rchisq(1, df = 1)
k <- 0
u <- runif(m)

for (i in 2:m) {
    xt <- x[i - 1]
    y <- rchisq(1, df = xt)
    num <- f(y, sigma) * dchisq(xt, df = y)
    den <- f(xt, sigma) * dchisq(y, df = xt)
    if (u[i] <= num/den)
        x[i] <- y else {
        x[i] <- xt
        k <- k + 1  #y is rejected
    }
}

k

## [1] 5297


reject <- 100 * (k/m)
reject

## [1] 52.97
```

```
index <- 5000:5500
y1 <- x[index]
par(mfrow = c(2, 1))
plot(x, type = "l", main = "", ylab = "x")
plot(index, y1, type = "l", main = "", ylab = "x")
```

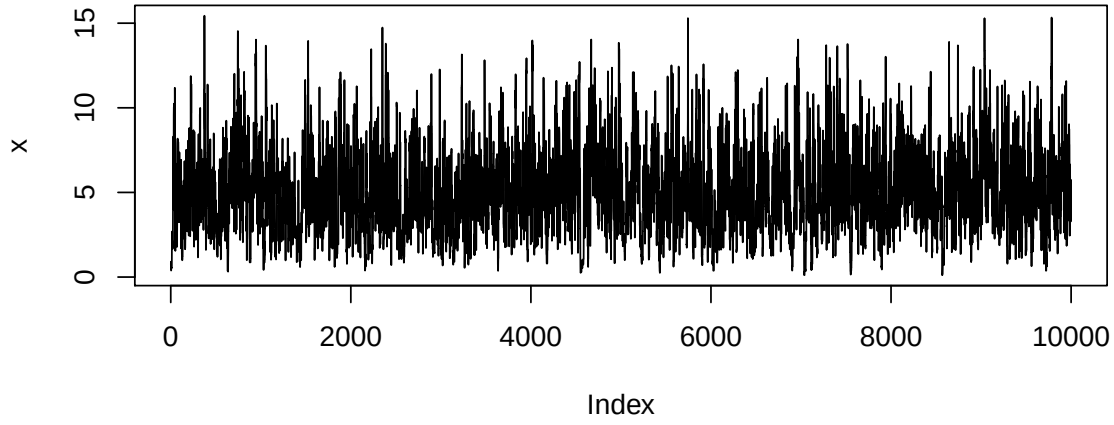


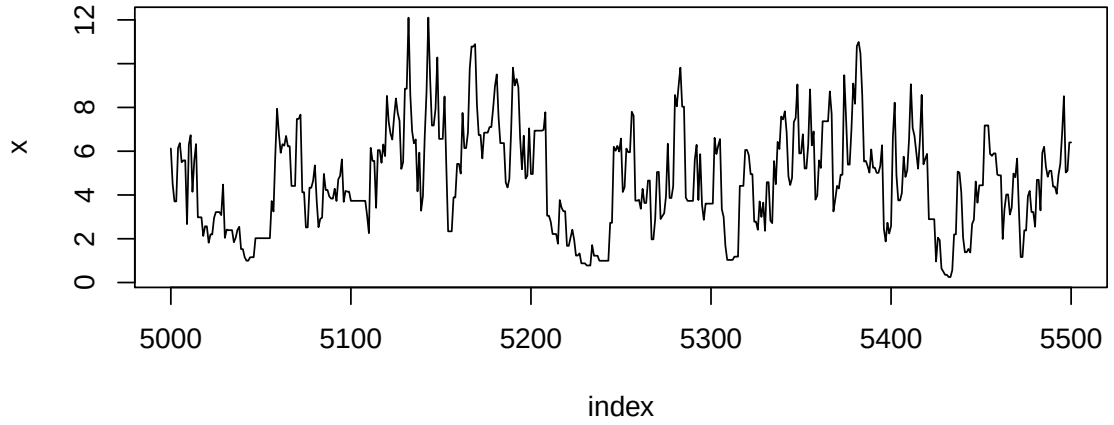

```
par(mfrow = c(1, 1))
```

Compared to example 9.1 this sampler is even more inefficient with a rejection rate of 53% as opposed to 40%. This higher rejection rate is clear in the zoomed in plot in which one can see that the value of x gets stuck for longer periods of time leading to poor mixing of the chain.

9.2 Repeat example 9.1 with gamma(X_t,1) proposal distribution.

```
set.seed(36)
f <- function(x, sigma) {
    if (any(x < 0))
        return(0)
    stopifnot(sigma > 0)
    return((x/sigma^2) * exp(-x^2/(2 * sigma^2)))
}

m <- 10000
sigma <- 4  #changed back to sigma=4
x <- numeric(m)
x[1] <- rgamma(1, shape = 1, rate = 1)
k <- 0
u <- runif(m)

for (i in 2:m) {
    xt <- x[i - 1]
    y <- rgamma(1, shape = xt, rate = 1)
    num <- f(y, sigma) * dgamma(xt, shape = y, rate = 1)
    den <- f(xt, sigma) * dgamma(y, shape = xt, rate = 1)
    if (u[i] <= num/den)
        x[i] <- y else {
        x[i] <- xt
        k <- k + 1  #y is rejected
    }
}

k

## [1] 2873


reject <- 100 * (k/m)
reject

## [1] 28.73


index <- 5000:5500
y1 <- x[index]
par(mfrow = c(2, 1))
plot(x, type = "l", main = "", ylab = "x")
plot(index, y1, type = "l", main = "", ylab = "x")
```






```
par(mfrow = c(1, 1))
```

This plot shows better mixing than than the Rayleigh with sigma=2 and has a lower rejection rate than the χ^2 version.

9.3 Useing Metropolis-Hastings sampler to generate a sntandard Cauchy

```
set.seed(36)
f <- function(x, theta, eta) {
    stopifnot(theta > 0)
    return(1/(theta * pi * (1 + ((x - eta)/theta)^2)))
}
```

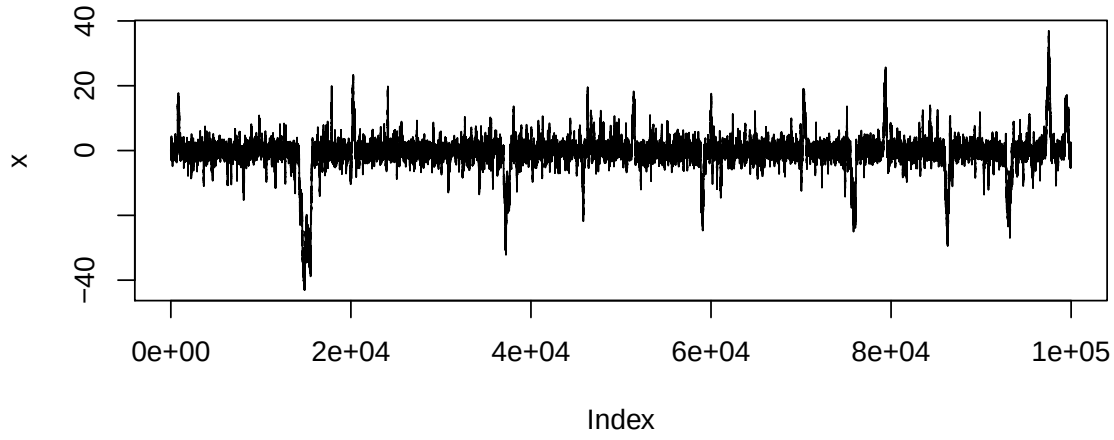
```r
m <- 1e+05
theta <- 1
eta <- 0
x <- numeric(m)
x[1] <- rnorm(1)  #use the standard normal to start.
k <- 0
u <- runif(m)
for (i in 2:m) {
    xt <- x[i - 1]
    y <- rnorm(1, mean = xt, sd = 1)
    num <- f(y, theta, eta)
    den <- f(xt, theta, eta)  # * dnorm(y, mean = xt, sd=1) it cancels out
    if (u[i] <= num/den)
        x[i] <- y else {
        x[i] <- xt
        k <- k + 1  #y is rejected
    }
}
k
```

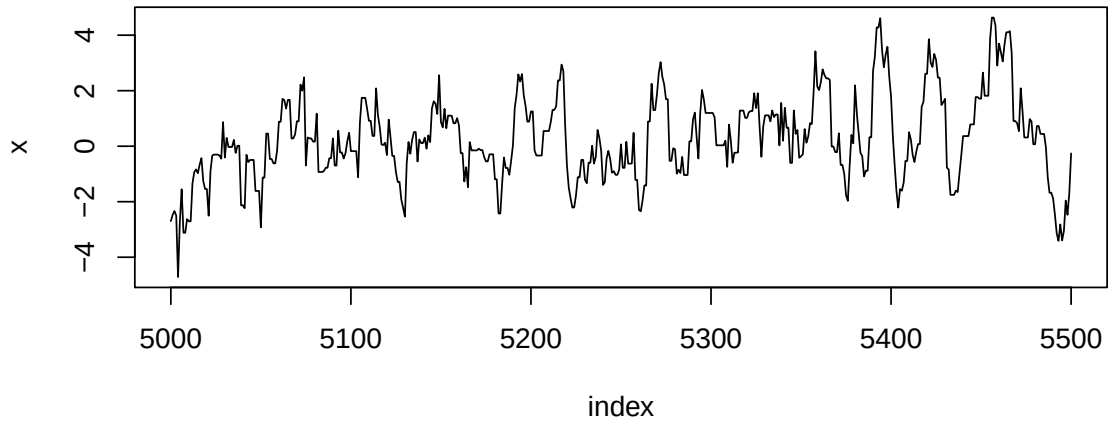
```
## [1] 22252
```

```r
reject <- 100 * (k/m)
reject
```

```
## [1] 22.25
```

```r
index <- 5000:5500
y1 <- x[index]
par(mfrow = c(2, 1))
plot(x, type = "l", main = "", ylab = "x")
plot(index, y1, type = "l", main = "", ylab = "x")
```





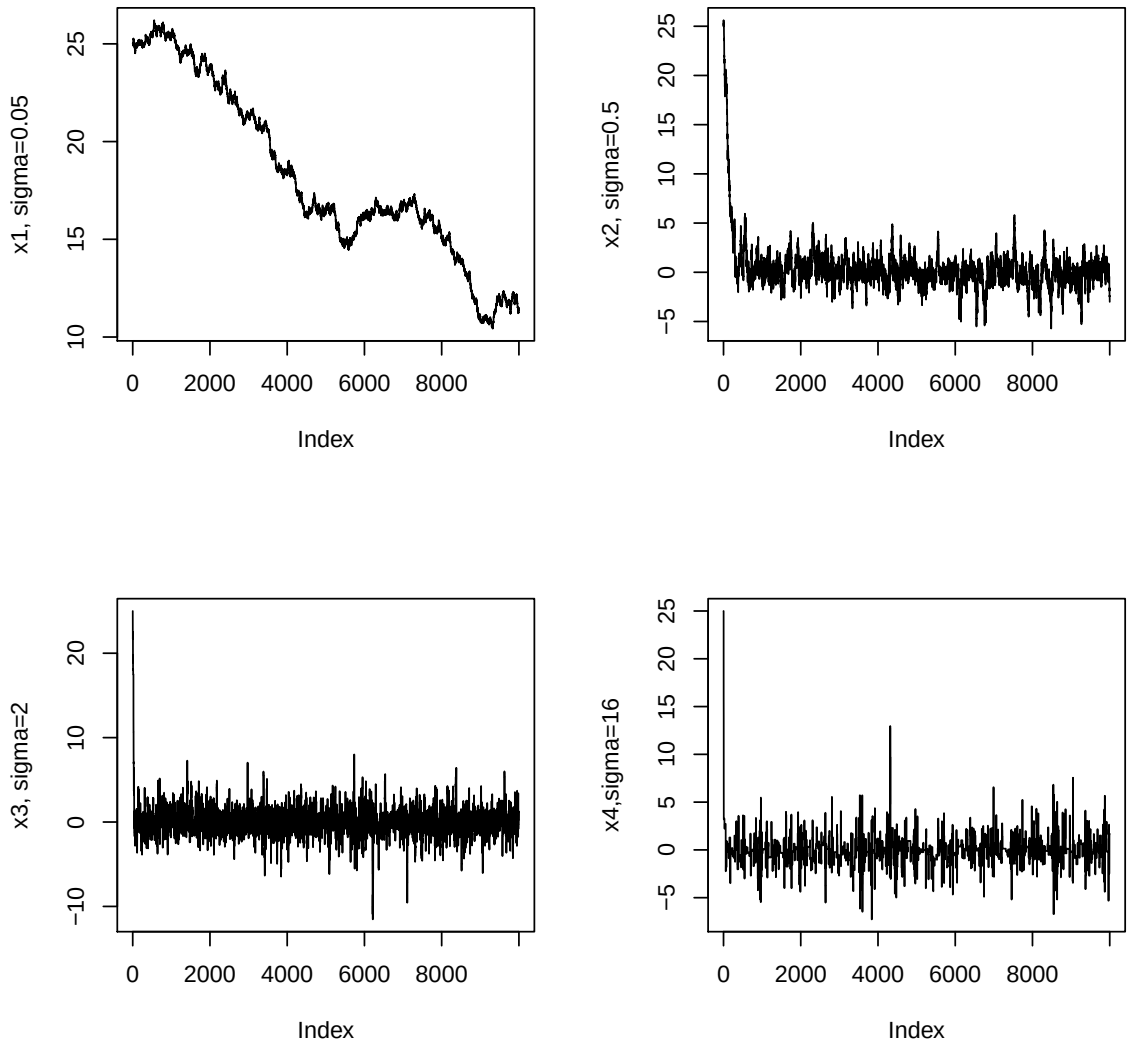

```
par(mfrow = c(1, 1))

c <- x[1001:length(x)]
q1 <- quantile(c, seq(0.1, 0.9, 0.1))
q2 <- qcauchy(seq(0.1, 0.9, 0.1))
q.compare <- rbind(q1, q2)
rownames(q.compare) <- c("From Sampler", "Theoretical")
```

I am not impressed with the approximation of this sample to the theoretical deciles.

9.4 Random walk Metropolis for Laplace distribution.

Using modified code from the book's website:

```r
set.seed(36)
dl <- function(x) {
    0.5 * exp(-abs(x))
}  #Standard laplace pdf

rw.Metropolis.l <- function(sigma, x0, N) {
    x <- numeric(N)
    x[1] <- x0
    u <- runif(N)
    k <- 0
    for (i in 2:N) {
        y <- rnorm(1, x[i - 1], sigma)
        if (u[i] <= (dl(y)/dl(x[i - 1])))
            x[i] <- y else {
            x[i] <- x[i - 1]
            k <- k + 1
        }
    }
    return(list(x = x, k = k))
}

N <- 10000
sigma <- c(0.05, 0.5, 2, 16)
x0 <- 25
rw1 <- rw.Metropolis.l(sigma[1], x0, N)
rw2 <- rw.Metropolis.l(sigma[2], x0, N)
rw3 <- rw.Metropolis.l(sigma[3], x0, N)
rw4 <- rw.Metropolis.l(sigma[4], x0, N)


par(mfrow = c(2, 2))
plot(rw1$x, type = "l", main = "", ylab = "x1, sigma=0.05")
plot(rw2$x, type = "l", main = "", ylab = "x2, sigma=0.5")
plot(rw3$x, type = "l", main = "", ylab = "x3, sigma=2")
plot(rw4$x, type = "l", main = "", ylab = "x4,sigma=16")
```




```
par(mfrow = c(1, 1))
```

```
# acceptnce rate
print(c(1 - (rw1$k/N), 1 - (rw2$k/N), 1 - (rw3$k/N), 1 - (rw4$k/N)))

## [1] 0.9815 0.8330 0.5291 0.0989
```

Only the chain with sigma=2 has the acceptance rate in the range recommended by the book. The mixing of the chain for sigma=2 also appears to tbe the best.

9.6 Posterior distribution of the multinomial distribution with probabilities,

$$\frac{\left(\frac{1}{2}+\frac{\theta}{4}, \frac{1-\theta}{4}, \frac{1-\theta}{4}, \frac{\theta}{4}\right)}{x_1!x_2!x_3!x_4!}$$

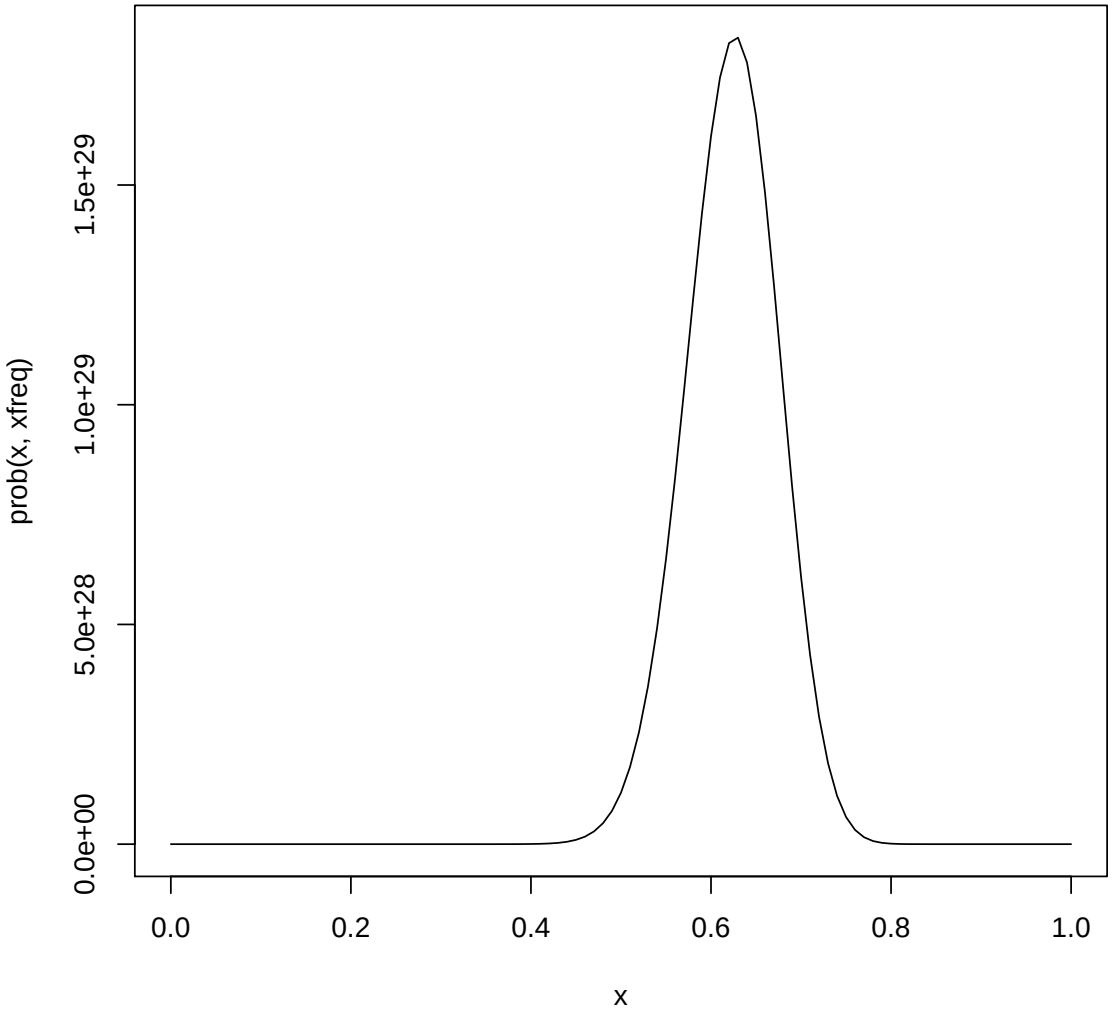
I will use the metropolis-hastings sampler with the uniform(0,1) as the prior (per request in the HW assignment) and implement with modified code from the book. The posterior density is proportional to:

$$f(X|\theta) = \frac{\pi(\theta)f(Y|\theta)}{f(Y)} \propto 1 \times p_1 \times p_2 \times p_3 \times p_4$$

which in this case is :

$$\left(\frac{2+Y}{4}\right)^{x_1}\left(\frac{1-Y}{4}\right)^{x_2+x_3}\left(\frac{Y}{4}\right)^{x_4}$$

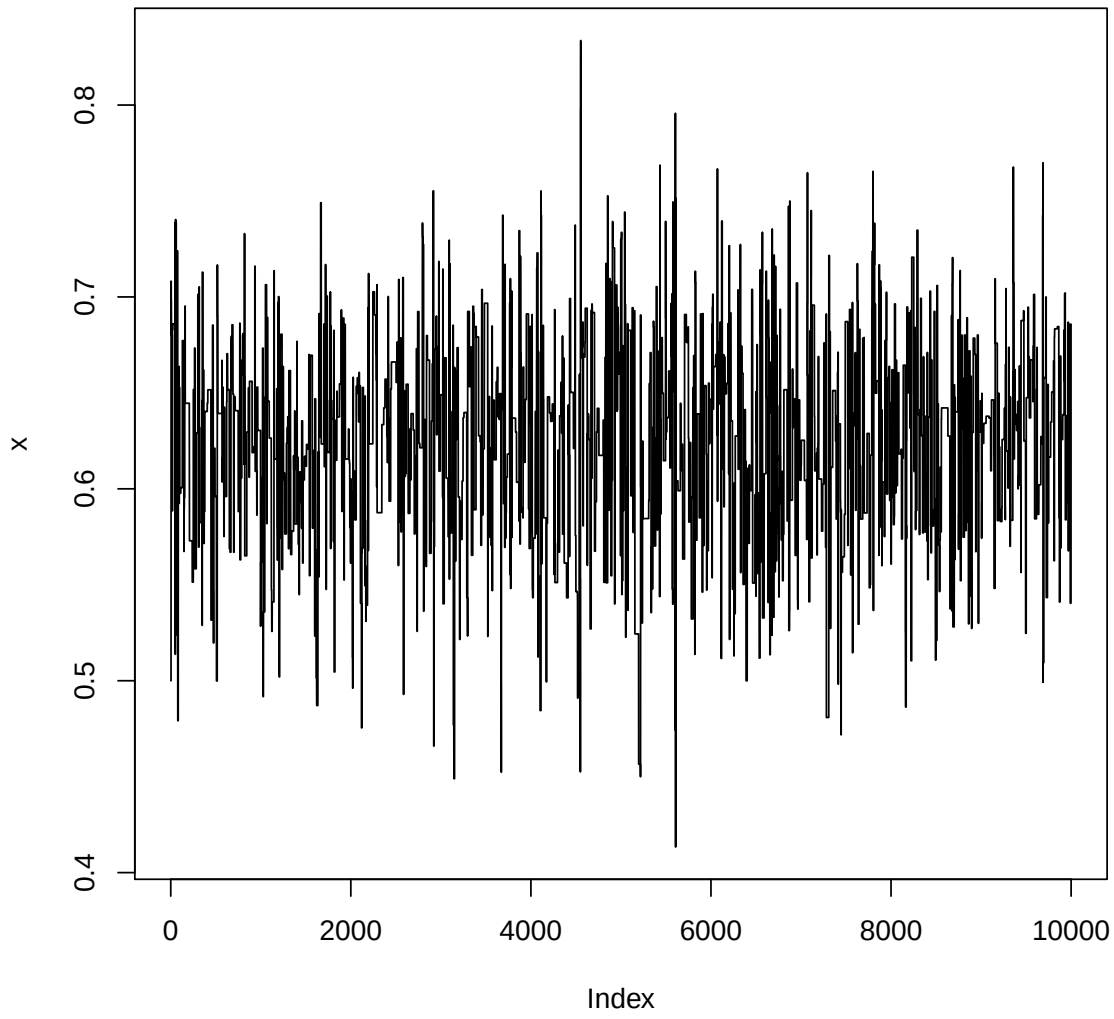
```
m <- 10000  #length of the chain

burn <- 1000  #burn-in time
animals <- 197
x <- numeric(m)  #the chain

# the observed frequencies of x
xfreq <- c(125, 18, 20, 34)

prob <- function(theta, x) {
    # computes (without the constants which cancel) the target density
    if (theta < 0 || theta > 1) {
        #bound on theta
        return(0)
    } else {
        p <- (2 + theta)^x[1] * (1 - theta)^(x[2] + x[3]) * (theta)^x[4]
        return(p)
    }
}
curve(prob(x, xfreq))
```



The Beta(α, β) has the same support as θ and I have frequently seen it used to model probabilities so I will use it as the proposal density. I experimented with various ways to set the parameters of the beta proposal density. Setting the two shape parameters equal to each other would create symmetric densities centered around .5. However, since the values of alpha and beta would be <1, the bulk of the dnsity would be close to 0 and 1, which led to a very low acceptance rate. Multiplying both alpha and beta by a constant to shift the density to the center did not alleviate the problem. Making the beta density assymetric by setting the beta=1 also lead to a very low acceptance rate. All of the methods arrived at a similar $\hat{\theta}$ estimate of

```
set.seed(36)
u <- runif(m) #for accept/reject step
k<-0
x[1] <- .5
```

```r
for (i in 2:m) {
    y <- rbeta(1, x[i-1], 1)  # symmetric proposal distribution

  if (u[i] <= (prob(y, xfreq)*dbeta(x[i-1],y,1)) / (prob(x[i-1], xfreq)*dbeta(y,x[i-1],1)))#
    x[i] <- y  else
        x[i] <- x[i-1]
        k <- k+1
}

plot( x, type="l", main="", ylab="x")
```



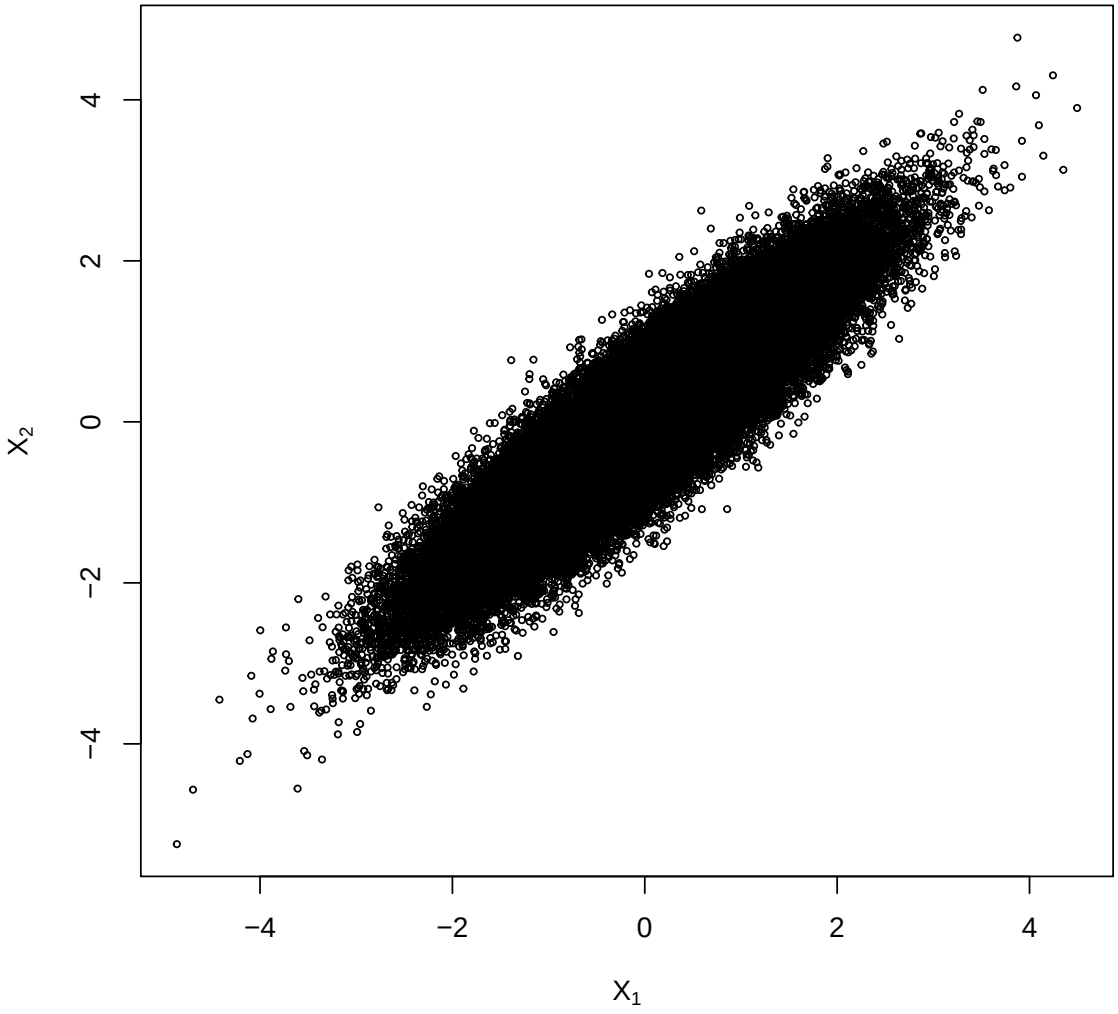
```
#acceptance rate
1-(k/m)

## [1] 1e-04


xb <- x[(burn+1):m]
th.hat<-mean(xb)
print(th.hat)

## [1] 0.6232


i <- sample(1:4, size=animals, replace=TRUE,prob=c((2+th.hat)/4,
                  (1-th.hat)/4, (1-th.hat)/4, th.hat/4))
predicted <- tabulate(i)
print(rbind(predicted,xfreq))

##           [,1] [,2] [,3] [,4]
## predicted  131   16   20   30
## xfreq      125   18   20   34
```

All of the methods arrived at a similar estimate of $\hat{\theta} = 0.62$. The predicted frequencies also matched up pretty well to the observed frequencies.

9.7 Bivariate normal via Gibbs sampling.
I will use code modified from the text.

```
set.seed(36)
# initialize constants and parameters
N <- 1e+05  #length of chain
burn <- 10000  #burn-in length
X <- matrix(0, N, 2)  #the chain, a bivariate sample

rho <- 0.9  #correlation
mu1 <- 0
mu2 <- 0
sigma1 <- 1
sigma2 <- 1
s1 <- sqrt(1 - rho^2) * sigma1
s2 <- sqrt(1 - rho^2) * sigma2

###### generate the chain #####

X[1, ] <- c(mu1, mu2)  #initialize

for (i in 2:N) {
    x2 <- X[i - 1, 2]
    m1 <- mu1 + rho * (x2 - mu2) * sigma1/sigma2
```

```
    X[i, 1] <- rnorm(1, m1, s1)
    x1 <- X[i, 1]
    m2 <- mu2 + rho * (x1 - mu1) * sigma2/sigma1
    X[i, 2] <- rnorm(1, m2, s2)
}

b <- burn + 1
x <- X[b:N, ]

# compare sample statistics to parameters
colMeans(x)

## [1] -0.004358 -0.003612

cov(x)

##         [,1]   [,2]
## [1,] 0.9830 0.8851
## [2,] 0.8851 0.9871

cor(x)

##         [,1]   [,2]
## [1,] 1.0000 0.8986
## [2,] 0.8986 1.0000


plot(x, main = "", cex = 0.5, xlab = bquote(X[1]), ylab = bquote(X[2]), ylim = range(x[,
    2]))
```



The sample statistics compare favorably with the parameters.

9.8 Gibbs sampler with Binomial and Beta conditional distributions.

The chain is generated by sampling from the two marginal densities; Binomial(n, y) and Beta$(x + a, n - x + b)$. In this case I will use a=2, b=3, and n=10.

```
set.seed(36)
# initialize constants and parameters
N <- 1e+05  #length of chain
burn <- 10000  #burn-in length
V <- matrix(0, N, 2)  #the chain, a bivariate sample
```

```
a <- 2  #correlation
b <- 3
n <- 10
y <- 0.5
x <- 0.5

###### generate the chain #####

V[1, ] <- c(y, x)  #initialize

for (i in 2:N) {
    y1 <- V[i - 1, 2]
    V[i, 1] <- rbinom(1, size = n, prob = y1)

    x1 <- V[i, 1]
    shape1 <- x1 + a
    shape2 <- n - x1 + b
    V[i, 2] <- rbeta(1, shape1, shape2)
}

b <- burn + 1
v <- V[b:N, ]

plot(v, main = "", cex = 0.5, xlab = y, ylab = x, ylim = range(x[, 2]))

## Error:  incorrect number of dimensions
```

9.9 Modification of the Gelmin-Rubin Convgergence monitor.

Using code from the book.

```
set.seed(36)
Gelman.Rubin <- function(psi) {
    # psi[i,j] is the statistic psi(X[i,1:j]) for chain in i-th row of X
    psi <- as.matrix(psi)
    n <- ncol(psi)
    k <- nrow(psi)
    psi.means <- rowMeans(psi)  #row means
    B <- n * var(psi.means)  #between variance est.
    psi.w <- apply(psi, 1, "var")  #within variances
    W <- mean(psi.w)  #within est.
    v.hat <- W * (n - 1)/n + (B/n)  #upper variance est.
    r.hat <- v.hat/W  #G-R statistic
    return(r.hat)
}

normal.chain <- function(sigma, N, X1) {
    # generates a Metropolis chain for Normal(0,1) with Normal(X[t], sigma)
    # proposal distribution and starting value X1
    x <- rep(0, N)
```

```
    x[1] <- X1
    u <- runif(N)

    for (i in 2:N) {
        xt <- x[i - 1]
        y <- rnorm(1, xt, sigma)  #candidate point
        r1 <- dnorm(y, 0, 1) * dnorm(xt, y, sigma)
        r2 <- dnorm(xt, 0, 1) * dnorm(y, xt, sigma)
        r <- r1/r2
        if (u[i] <= r)
            x[i] <- y else x[i] <- xt
    }
    return(x)
}

sigma <- 0.2  #parameter of proposal distribution
k <- 4  #number of chains to generate
n <- 15000  #length of chains
b <- 1000  #burn-in length

# choose overdispersed initial values
x0 <- c(-10, -5, 5, 10)
# generate the chains
X <- matrix(0, nrow = k, ncol = n)
for (i in 1:k) X[i, ] <- normal.chain(sigma, n, x0[i])

# compute diagnostic statistics
psi <- t(apply(X, 1, cumsum))
for (i in 1:nrow(psi)) psi[i, ] <- psi[i, ]/(1:ncol(psi))
print(Gelman.Rubin(psi))

## [1] 1.172
```