**Problem 1-4**

**(a)**

rw.Metropolis <- function(n, sigma, x0, N, k, dis, b){

x <- numeric(N)

p <- numeric(N-1)

x[1] <- x0

u <- runif(N)

f <- 0

for (i in 2:N){

if(dis == 0){

y <- rnorm(1, x[i-1], sigma)

} else if (dis == 1){

y <- runif(1, x[i-1]-sigma, x[i-1]+sigma)

}

p[i-1] <- min(dt(y,n) / dt(x[i-1], n), 1)

if(u[i] <= p[i-1]){

x[i] <- y

} else {

x[i] <- x[i-1]

if(i >= b){

f <- f+1

}

}

}

x <- x[b:N]

p <- p[b:(N-1)]

#autocorrelation

ac <- numeric(k)

a <- mean(x)

den <- sum((x-a)^2)

for(i in 1:k){

nu <- 0.0

for(j in 1:(N-b+1-i)){

nu <- nu + (x[j] - a)\*(x[j+i] - a)

}

ac[i] <- nu/den

}

return(list(x=x, f=f/(N-b+1), p=p, ac=ac))

}

set.seed(1000)

n <- 5

N <- 25000

k <- 50

b <- 5001

sigma <- c(1, sqrt(.6), sqrt(.2), 1)

x0 <- 25

rw1 <- rw.Metropolis(n, sigma[1], x0, N, k, 0, b)

rw2 <- rw.Metropolis(n, sigma[2], x0, N, k, 0, b)

rw3 <- rw.Metropolis(n, sigma[3], x0, N, k, 0, b)

rw4 <- rw.Metropolis(n, sigma[4], x0, N, k, 1, b)

#Mean

avg <- c(mean(rw1$x), mean(rw2$x), mean(rw3$x), mean(rw4$x))

print(avg)

#Acceptance Rate

ar <- c(mean(rw1$p), mean(rw2$p), mean(rw3$p), mean(rw4$p))

print(ar)

#Rejection Rate

print(c(rw1$f, rw2$f, rw3$f, rw4$f))

#Autocorrelation

attach(mtcars)

par(mfrow=c(2,2))

plot(rw1$ac, main="norm, mean = 0, var = 1", xlab="lag", ylab="ACF", ylim=c(0,1))

plot(rw2$ac, main="norm, mean = 0, var = 0.6", xlab="lag", ylab="ACF", ylim=c(0,1))

plot(rw3$ac, main="norm, mean = 0, var = 0.2", xlab="lag", ylab="ACF", ylim=c(0,1))

plot(rw4$ac, main="unif, -1, 1", xlab="lag", ylab="ACF", ylim=c(0,1))

#Quantiles Q-Q plot

a <- ppoints(100)

QT <- qt(a, n)

Q1 <- quantile(rw1$x, a)

Q2 <- quantile(rw2$x, a)

Q3 <- quantile(rw3$x, a)

Q4 <- quantile(rw4$x, a)

frame()

attach(mtcars)

par(mfrow=c(2,2))

qqplot(QT, Q1, main="norm, mean = 0, var = 1",

xlab="t Quantiles", ylab="Sample Quantiles")

qqplot(QT, Q1, main="norm, mean = 0, var = 0.6",

xlab="t Quantiles", ylab="Sample Quantiles")

qqplot(QT, Q1, main="norm, mean = 0, var = 0.2",

xlab="t Quantiles", ylab="Sample Quantiles")

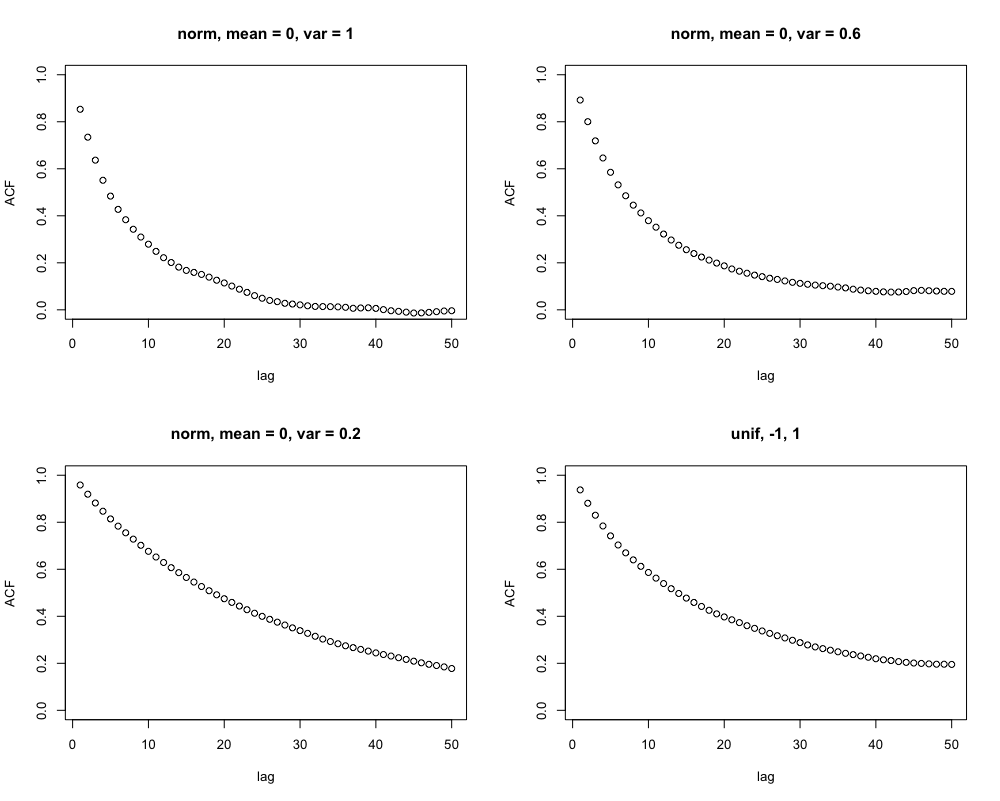
qqplot(QT, Q1, main="unif, -1, 1",

xlab="t Quantiles", ylab="Sample Quantiles")

**(b)** The Monte Carlo estimates of the mean of this Student’s t distribution given by 4 proposal densities are **0.03290480, 0.08758312, 0.03092236, and -0.08668746**. Since the Student’s t distribution should have a theoretical mean of 0, our estimates are close enough.

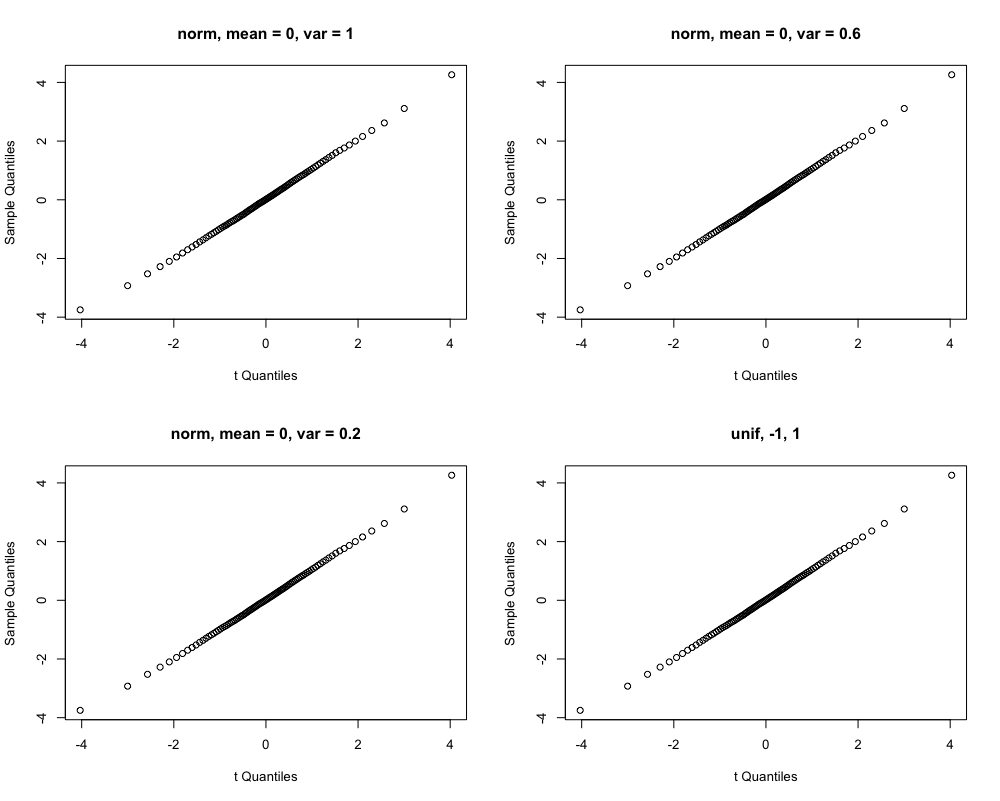
**(c)** The acceptance rates are **0.7209302, 0.7776595, 0.8680405, and 0.8199232**, while the rejection rates (the frequencies that new Y got rejected) are 0.27700, 0.22405, 0.12985, and 0.17965.

**(d)** Plot of sample autocorrelation function for first 50 lags:



We can see the simulations were working well because the kth lag autocorrelation always became smaller as k increased.

**(e)** Q-Q plot of the MCMC sample quantiles against the theoretical quantiles of the Student’s t distribution:



We can see the simulations are working correctly because our sample quantiles appropriately matched the theoretical quantiles.

**Problem 5**

We could use a variance of , where d is the number of dimensions of our target. For example, if d = 10, we should set and thus have an average acceptance rate of around 0.230. Although d should theoretically be infinite, this would work for finite-dimensional situations for d as small as 5. This does not work well for smaller d. For example, if d = 1, the optimal acceptance rate becomes 0.44.

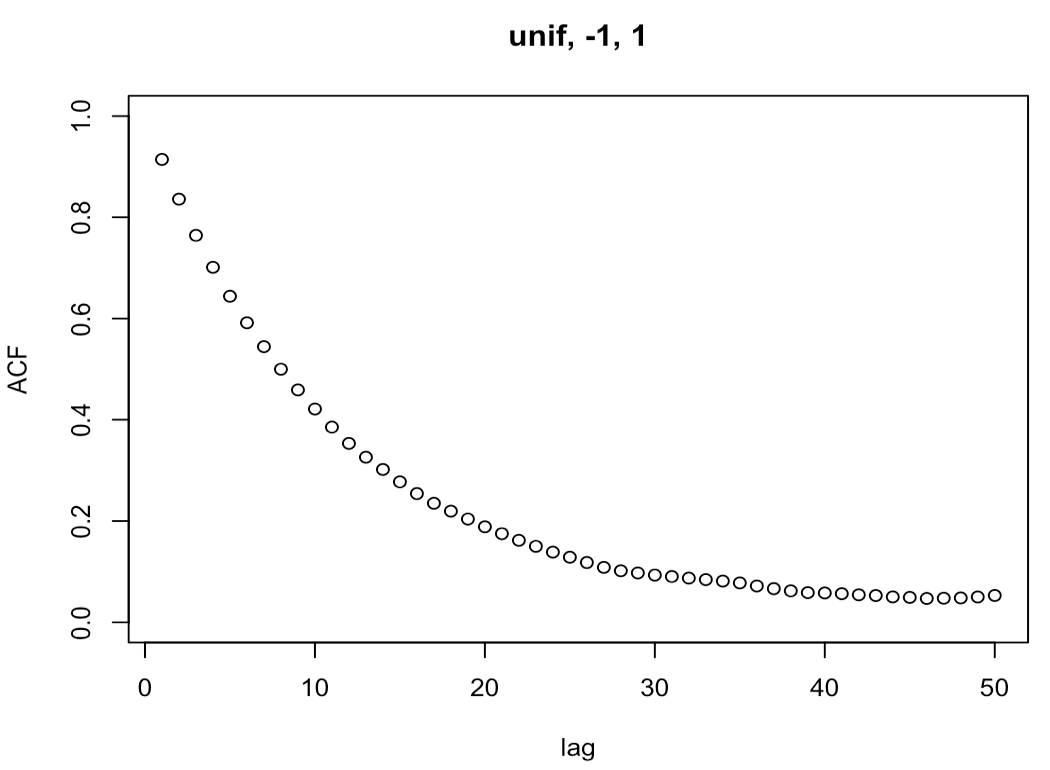
**Problem 6**

Since the simulation in problem 4 had the greatest problem with high autocorrelation, we try to tune it.

Firstly, lengthen burn-in period to 10000:

rw4 <- rw.Metropolis(n, sigma[4], x0, N, k, 1, b+5000)

plot(rw4$ac, main="unif, -1, 1", xlab="lag", ylab="ACF", ylim=c(0,1))



This seems to have an effect, as the autocorrelation became even smaller with higher k, although the magnitude of change is small.

Secondly, use thinning and keep only every 10th value:

rw.MetropolisThin <- function(n, sigma, x0, N, k, dis, b, t){

x <- numeric(N)

p <- numeric(N-1)

x[1] <- x0

u <- runif(N)

f <- 0

for (i in 2:N){

if(dis == 0){

y <- rnorm(1, x[i-1], sigma)

} else if (dis == 1){

y <- runif(1, x[i-1]-sigma, x[i-1]+sigma)

}

p[i-1] <- min(dt(y,n) / dt(x[i-1], n), 1)

if(u[i] <= p[i-1]){

x[i] <- y

} else {

x[i] <- x[i-1]

}

}

x <- x[b:N]

x <- x[seq(1, length(x), t)]

p <- p[b:(N-1)]

p <- p[seq(1, length(p), t)]

#autocorrelation

ac <- numeric(k)

a <- mean(x)

den <- sum((x-a)^2)

for(i in 1:k){

nu <- 0.0

for(j in 1:(length(x)-i)){

nu <- nu + (x[j] - a)\*(x[j+i] - a)

}

ac[i] <- nu/den

}

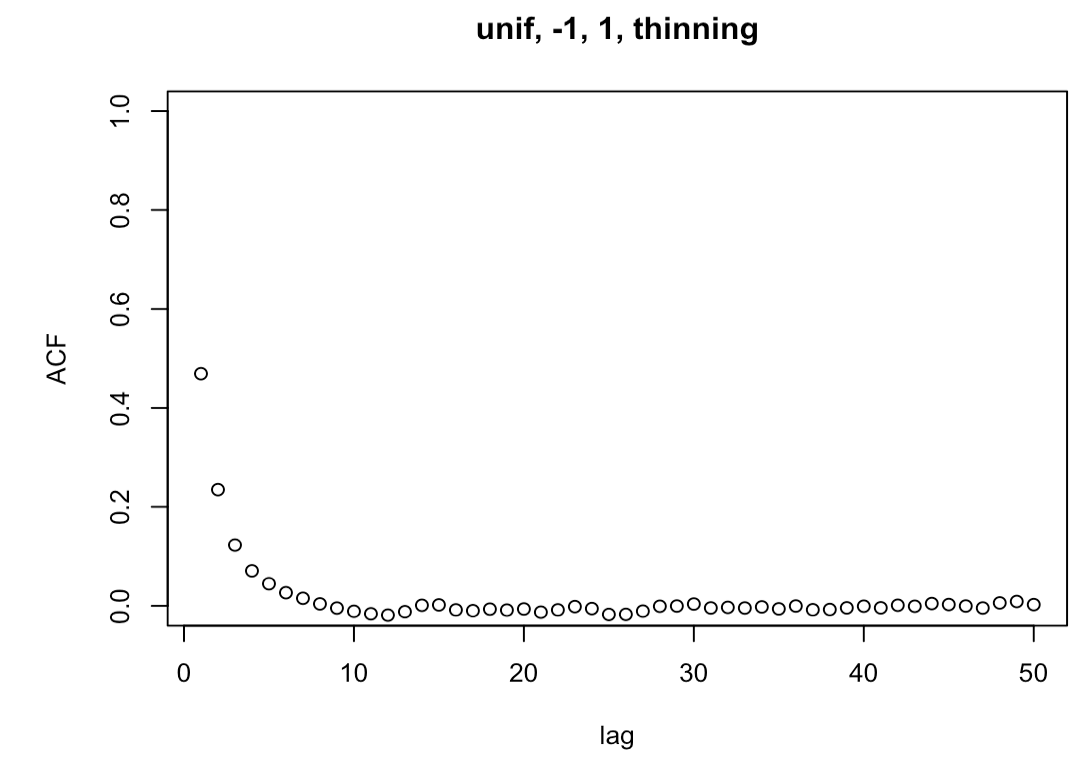
return(list(x=x, p=p, ac=ac))

}

t <- 10

rw4 <- rw.MetropolisThin(n, sigma[4], x0, N\*t, k, 1, b, t)

plot(rw4$ac, main="unif, -1, 1, thinning", xlab="lag", ylab="ACF", ylim=c(0,1))



As we can see, the thinning does have an effect, and the autocorrelation is significantly reduced with higher k. However, as we learned in class, this is unnecessary and inefficient.

**Problem 7**

**(a)**

set.seed(1000)

N <- 25000

burn <- 5001

n <- 16

a <- 2

b <- 4

x0 <- 10

y0 <- .5

X <- matrix(0, N, 2)

X[1,] <- c(x0, y0)

for(i in 2:N){

X[i, 1] <- rbinom(1, n, X[i-1, 2])

X[i, 2] <- rbeta(1, X[i-1, 1] + a, n - X[i-1, 1] + b)

}

b <- burn + 1

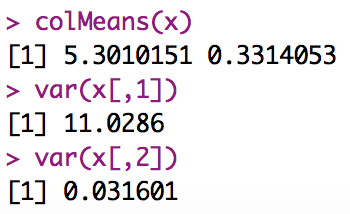
x <- X[b:N,]

colMeans(x)

var(x[,1])

var(x[,2])

**(b)** Means and variances are estimated as below:



**(c)** We could do this by firstly draw from a beta distribution to determine the p for binomial distribution, and then draw from that binomial distribution to get x:

y <- rbeta(20000,2,4)

x2 <- numeric(20000)

for(i in 1:20000){

x2[i] <- rbinom(1,16,y[i])

}

mean(x2)

var(x2)

This method gives us an estimate of mean = 5.3614 and variance = 11.06624, which is close to what we got from Gibbs algorithm.

**(d)**

a <- ppoints(100)

Q1 <- quantile(x[,1], a)

Q2 <- quantile(x2, a)

qqplot(Q1, Q2, main="", xlab="(a) Quantiles", ylab="(c) Quantiles")

