Lab 4 Metropolis Hastings

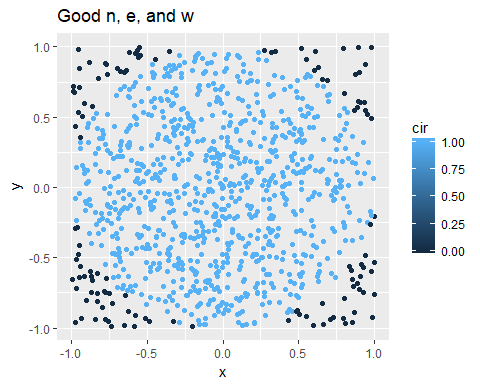
Adam Hinthorne

April 5, 2018

# 1.a.

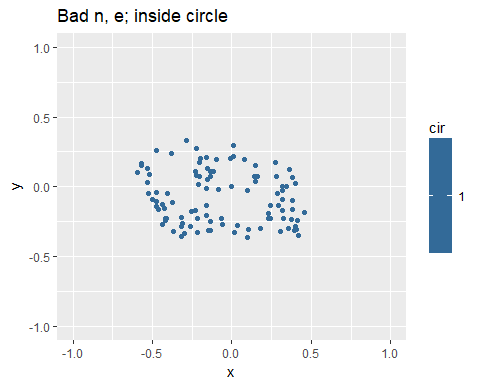
# Naive Start: Random Walk without Metropolis Hastings (i.e. Always moving)

x<-c(0)  
y<-c(0)  
w<-x  
z<-y  
e<-1  
n<-c(1000)  
  
set.seed(47)  
  
#Basic  
for (i in 2:n){  
 while (w==x[i-1]){  
 w<-c()  
 w<-runif(1,x[i-1]-e,x[i-1]+e)  
 w<-ifelse(w>1 | w<(-1),x[i-1],w)  
 }  
x[i]<-w  
while (z==y[i-1]){  
 z<-c()  
 z<-runif(1,y[i-1]-e,y[i-1]+e)  
 z<-ifelse(z>1 | z<(-1),y[i-1],z)  
}  
y[i]<-z  
}  
  
points<-as.data.frame(cbind(x,y))  
cir<-c()  
for (i in 1:length(x)){  
cir[i]<-ifelse(1>sqrt((x[i]^2+y[i]^2)),1,0)  
}  
points<-as.data.frame(cbind(points,cir))  
  
ggplot(points,aes(x=x,y=y,col=cir))+geom\_jitter()+ggtitle("Good n, e, and w")



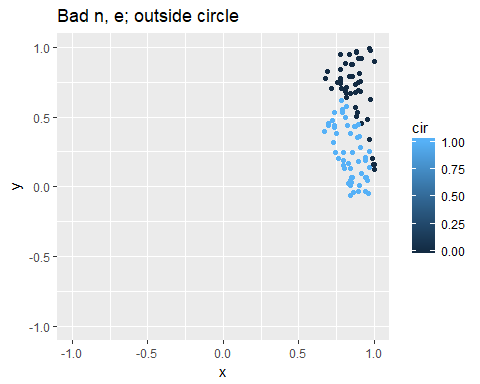
The problem with moving every single time is that you are dependent on where you start, how big your error is, and how long the length of your run is. For example a small error, with a short run period results in particularly bad estimations. In the next plot we see that we overestimate the number of points in the circle if we start in the circle and underestimate the true proportion of points in the circle if we start outside the circle. We also need to use the entire unit circle, otherwise we might get stuck on the x and y axis resulting in an inaccurate representation.

x<-c(0)  
y<-c(0)  
w<-x  
z<-y  
e<-.1  
n<-c(100)  
  
set.seed(47)  
  
#Basic  
for (i in 2:n){  
 while (w==x[i-1]){  
 w<-c()  
 w<-runif(1,x[i-1]-e,x[i-1]+e)  
 w<-ifelse(w>1 | w<(-1),x[i-1],w)  
 }  
x[i]<-w  
while (z==y[i-1]){  
 z<-c()  
 z<-runif(1,y[i-1]-e,y[i-1]+e)  
 z<-ifelse(z>1 | z<(-1),y[i-1],z)  
}  
y[i]<-z  
}  
  
points<-as.data.frame(cbind(x,y))  
cir<-c()  
for (i in 1:length(x)){  
cir[i]<-ifelse(1>sqrt((x[i]^2+y[i]^2)),1,0)  
}  
points<-as.data.frame(cbind(points,cir))  
  
ggplot(points,aes(x=x,y=y,col=cir))+geom\_jitter()+ggtitle("Bad n, e; inside circle")+xlim(-1,1)+ylim(-1,1)

 As we can see all of these points are in the circle, which is not an accurate representation.

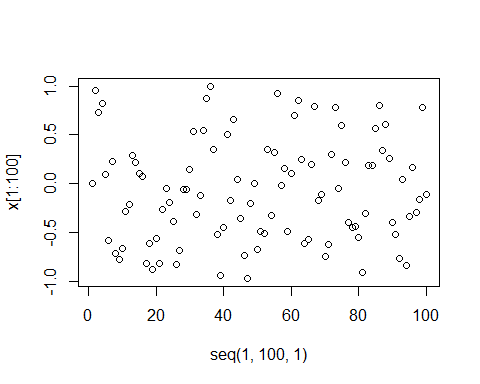
x<-c(1)  
y<-c(1)  
w<-x  
z<-y  
e<-.1  
n<-c(100)  
  
set.seed(47)  
  
#Basic  
for (i in 2:n){  
 while (w==x[i-1]){  
 w<-c()  
 w<-runif(1,x[i-1]-e,x[i-1]+e)  
 w<-ifelse(w>1 | w<(-1),x[i-1],w)  
 }  
x[i]<-w  
while (z==y[i-1]){  
 z<-c()  
 z<-runif(1,y[i-1]-e,y[i-1]+e)  
 z<-ifelse(z>1 | z<(-1),y[i-1],z)  
}  
y[i]<-z  
}  
  
points<-as.data.frame(cbind(x,y))  
cir<-c()  
for (i in 1:length(x)){  
cir[i]<-ifelse(1>sqrt((x[i]^2+y[i]^2)),1,0)  
}  
points<-as.data.frame(cbind(points,cir))  
  
ggplot(points,aes(x=x,y=y,col=cir))+geom\_jitter()+ggtitle("Bad n, e; outside circle")+xlim(-1,1)+ylim(-1,1)

## Warning: Removed 1 rows containing missing values (geom\_point).

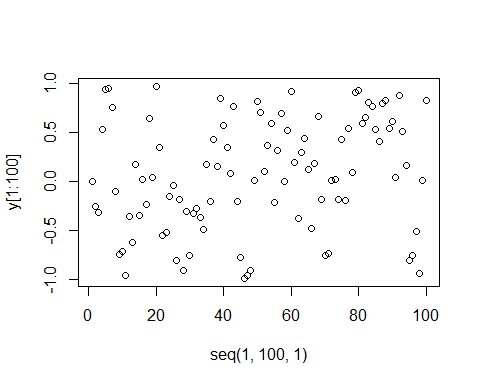
 Now we see that the proportion is off the other way underestimating the

# 1. b. Naive estimator without metropolis hastings:

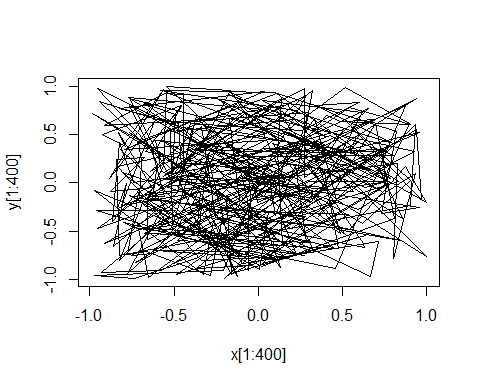
x<-c(0)  
y<-c(0)  
w<-x  
z<-y  
e<-1  
n<-c(1000)  
  
set.seed(47)  
  
#Basic  
for (i in 2:n){  
 while (w==x[i-1]){  
 w<-c()  
 w<-runif(1,x[i-1]-e,x[i-1]+e)  
 w<-ifelse(w>1 | w<(-1),x[i-1],w)  
 }  
x[i]<-w  
while (z==y[i-1]){  
 z<-c()  
 z<-runif(1,y[i-1]-e,y[i-1]+e)  
 z<-ifelse(z>1 | z<(-1),y[i-1],z)  
}  
y[i]<-z  
}  
  
plot(x=seq(1,100,1),y=x[1:100])



plot(x=seq(1,100,1),y=y[1:100])



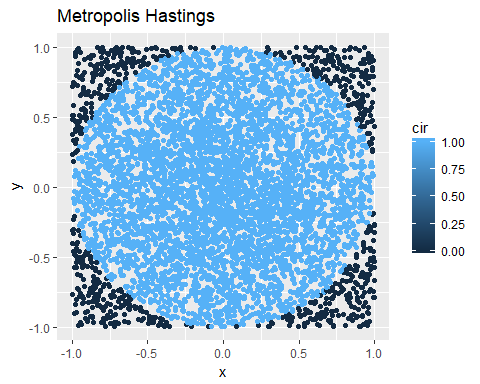
plot(x=x[1:400],y=y[1:400],type="l")



From these plots we know that the Naive random walk moves the points every single time, making it especially sensitive to changes in the error, starting point, and length of the run.

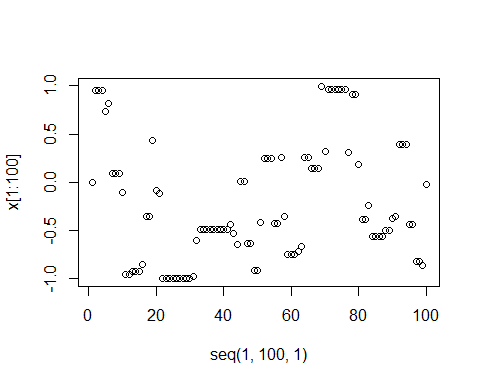
# 1.a. (again) Metropolis Hastings

x<-c(0)  
y<-c(0)  
w<-x  
z<-y  
e<-1  
n<-c(10000)  
  
set.seed(47)  
  
for (i in 2:n){  
 w<-c()  
 w<-runif(1,x[i-1]-e,x[i-1]+e)  
 r<-dunif(w,-1,1)\*dunif(x[i-1],w-e,w+e)/(dunif(x[i-1],-1,1)\*dunif(w,x[i-1]-e,x[i-1]+e))  
 x[i]<-ifelse(min(1,r),w,x[i-1])  
   
 z<-c()  
 z<-runif(1,y[i-1]-e,y[i-1]+e)   
 rt<-dunif(z,-1,1)\*dunif(y[i-1],z-e,z+e)/(dunif(y[i-1],-1,1)\*dunif(z,y[i-1]-e,y[i-1]+e))  
 y[i]<-ifelse(min(1,rt),z,y[i-1])  
   
 x[i]<-ifelse(x[i]==x[i-1] | y[i]==y[i-1],x[i-1],x[i])  
 y[i]<-ifelse(x[i]==x[i-1] | y[i]==y[i-1],y[i-1],y[i])  
}  
  
points<-as.data.frame(cbind(x,y))  
cir<-c()  
for (i in 1:length(x)){  
cir[i]<-ifelse(1>sqrt((x[i]^2+y[i]^2)),1,0)  
}  
points<-as.data.frame(cbind(points,cir))  
  
ggplot(points,aes(x=x,y=y,col=cir))+geom\_jitter()+ggtitle("Metropolis Hastings")

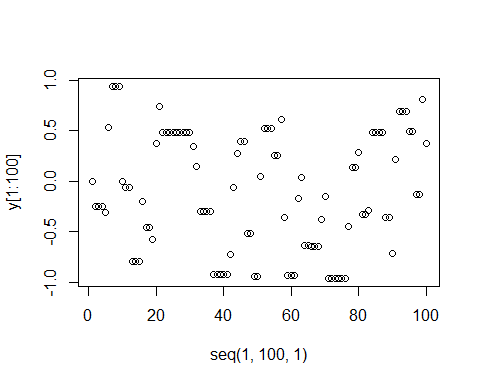
 Using metropolis hastings we see that we can account for the problems we saw with the naive random walk. The starting point matters less because we move towards the center of the circle with a higher probability if we are near the edge and we move outside the circle with a reasonable probability in the circle. In addition the we can more easily calibrate the length of the run as well as the error term to compensate for eachother and get an accurate estimation.

# b.

x<-c(0)  
y<-c(0)  
w<-x  
z<-y  
e<-1  
n<-c(10000)  
  
set.seed(47)  
  
for (i in 2:n){  
 w<-c()  
 w<-runif(1,x[i-1]-e,x[i-1]+e)  
 r<-dunif(w,-1,1)\*dunif(x[i-1],w-e,w+e)/(dunif(x[i-1],-1,1)\*dunif(w,x[i-1]-e,x[i-1]+e))  
 x[i]<-ifelse(min(1,r),w,x[i-1])  
   
 z<-c()  
 z<-runif(1,y[i-1]-e,y[i-1]+e)   
 rt<-dunif(z,-1,1)\*dunif(y[i-1],z-e,z+e)/(dunif(y[i-1],-1,1)\*dunif(z,y[i-1]-e,y[i-1]+e))  
 y[i]<-ifelse(min(1,rt),z,y[i-1])  
   
 x[i]<-ifelse(x[i]==x[i-1] | y[i]==y[i-1],x[i-1],x[i])  
 y[i]<-ifelse(x[i]==x[i-1] | y[i]==y[i-1],y[i-1],y[i])  
}  
  
plot(x=seq(1,100,1),y=x[1:100])



plot(x=seq(1,100,1),y=y[1:100])

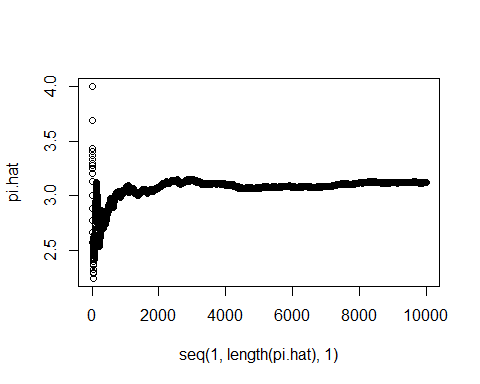
 As we can see the metropolis hastings algorithm causes the points to move less, and stay in the same place if the M-H ratio is low.

# c.

x<-c(0)  
y<-c(0)  
w<-x  
z<-y  
e<-1  
n<-c(10000)  
  
set.seed(47)  
  
for (i in 2:n){  
 w<-c()  
 w<-runif(1,x[i-1]-e,x[i-1]+e)  
 r<-dunif(w,-1,1)\*dunif(x[i-1],w-e,w+e)/(dunif(x[i-1],-1,1)\*dunif(w,x[i-1]-e,x[i-1]+e))  
 x[i]<-ifelse(min(1,r),w,x[i-1])  
   
 z<-c()  
 z<-runif(1,y[i-1]-e,y[i-1]+e)   
 rt<-dunif(z,-1,1)\*dunif(y[i-1],z-e,z+e)/(dunif(y[i-1],-1,1)\*dunif(z,y[i-1]-e,y[i-1]+e))  
 y[i]<-ifelse(min(1,rt),z,y[i-1])  
   
 x[i]<-ifelse(x[i]==x[i-1] | y[i]==y[i-1],x[i-1],x[i])  
 y[i]<-ifelse(x[i]==x[i-1] | y[i]==y[i-1],y[i-1],y[i])  
}  
  
points<-as.data.frame(cbind(x,y))  
cir<-c()  
for (i in 1:length(x)){  
cir[i]<-ifelse(1>sqrt((x[i]^2+y[i]^2)),1,0)  
}  
points<-as.data.frame(cbind(points,cir))  
  
pi.hat<-c()  
for(i in 1:length(x)){  
pi.hat[i]<-4\*sum(points$cir[1:i])/(i)  
}  
4\*(sum(points$cir))/(length(points$cir))

## [1] 3.1208

plot(x=seq(1,length(pi.hat),1),pi.hat)



As we can see we get a reasonable estimation of with the M-H algoritm, and it approaches pie as n increases.

# d. Burn in

As we see from the plot in part c, it actually takes awhile for the algorithm to reach the correct estimation of . This could be due to randomness in whether points move or not and there may be periods when the algorithm gets stuck at the edges. The “burn in” period accounts for this, allowing a critical mass of points to be created for both classes before actuallly calculating them for the total.

# 2.

#good prior  
  
alpha <-4  
beta <-1  
  
x <- c()  
  
#start with random x0  
x[1] <- rbeta(1, alpha, beta)  
  
for (i in 1:1000) {  
 #randomly choose y  
 y <- rbeta(1, alpha, beta)  
   
 #compute MH ratio  
 a <- dbeta(y, alpha + x[i], beta + 1 - x[i])  
 b <- dbeta(x[i], alpha + x[i], beta + 1 - x[i])  
 c <- dbeta(x[i], alpha, beta)  
 d <- dbeta(y, alpha, beta)  
 MH <- (a\*c)/(b\*d)  
   
 #now figure out if you move or not  
 u <- runif(1,0,1)  
   
 if (MH > u) {  
 x[i + 1] <- y  
 }  
   
 if (MH < u) {  
 x[i + 1] <- x[i]  
 }  
   
}  
  
mean(x)

## [1] 0.7904923

#check how accurate this is  
beta\_mean <- alpha/(alpha + beta)  
beta\_mean

## [1] 0.8

With a good prior we reach a reasonable estimation using the M-H algorithm.

alpha <-.09  
beta <-.4  
  
x <- c()  
  
#start with random x0  
x[1] <- rbeta(1, alpha, beta)  
  
for (i in 1:1000) {  
 #randomly choose y  
 y <- rbeta(1, alpha, beta)  
   
 #compute MH ratio  
 a <- dbeta(y, alpha + x[i], beta + 1 - x[i])  
 b <- dbeta(x[i], alpha + x[i], beta + 1 - x[i])  
 c <- dbeta(x[i], alpha, beta)  
 d <- dbeta(y, alpha, beta)  
 MH <- (a\*c)/(b\*d)  
   
 #now figure out if you move or not  
 u <- runif(1,0,1)  
   
 if (MH > u) {  
 x[i + 1] <- y  
 }  
   
 if (MH < u) {  
 x[i + 1] <- x[i]  
 }  
   
}  
  
mean(x)

## [1] 0.300758

#check how accurate this is  
beta\_mean <- alpha/(alpha + beta)  
beta\_mean

## [1] 0.1836735

However with a bad prior, we miss an accurate estimation of the mean of the beta distribution.