Module 4 HW

Yichen Dong

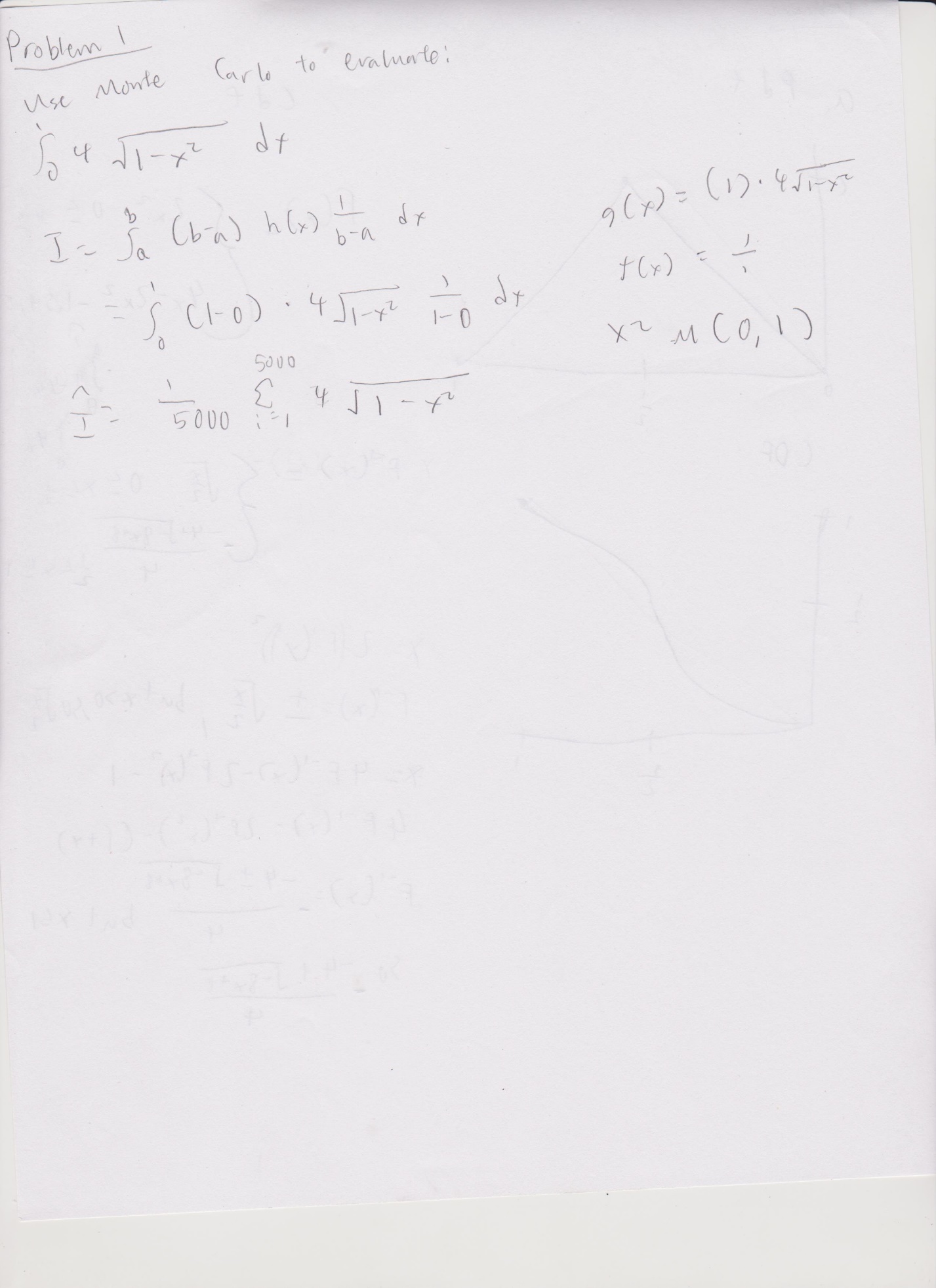
September 27, 2018

## Problem 1

set.seed(121)  
x = runif(5000,0,1)  
montycarlo = data.frame(x)  
montycarlo = montycarlo %>%   
 mutate(g\_x = 4\*sqrt(1-x^2))  
  
I\_hat = with(montycarlo, 1/length(montycarlo$x)\*sum(g\_x))  
I\_hat

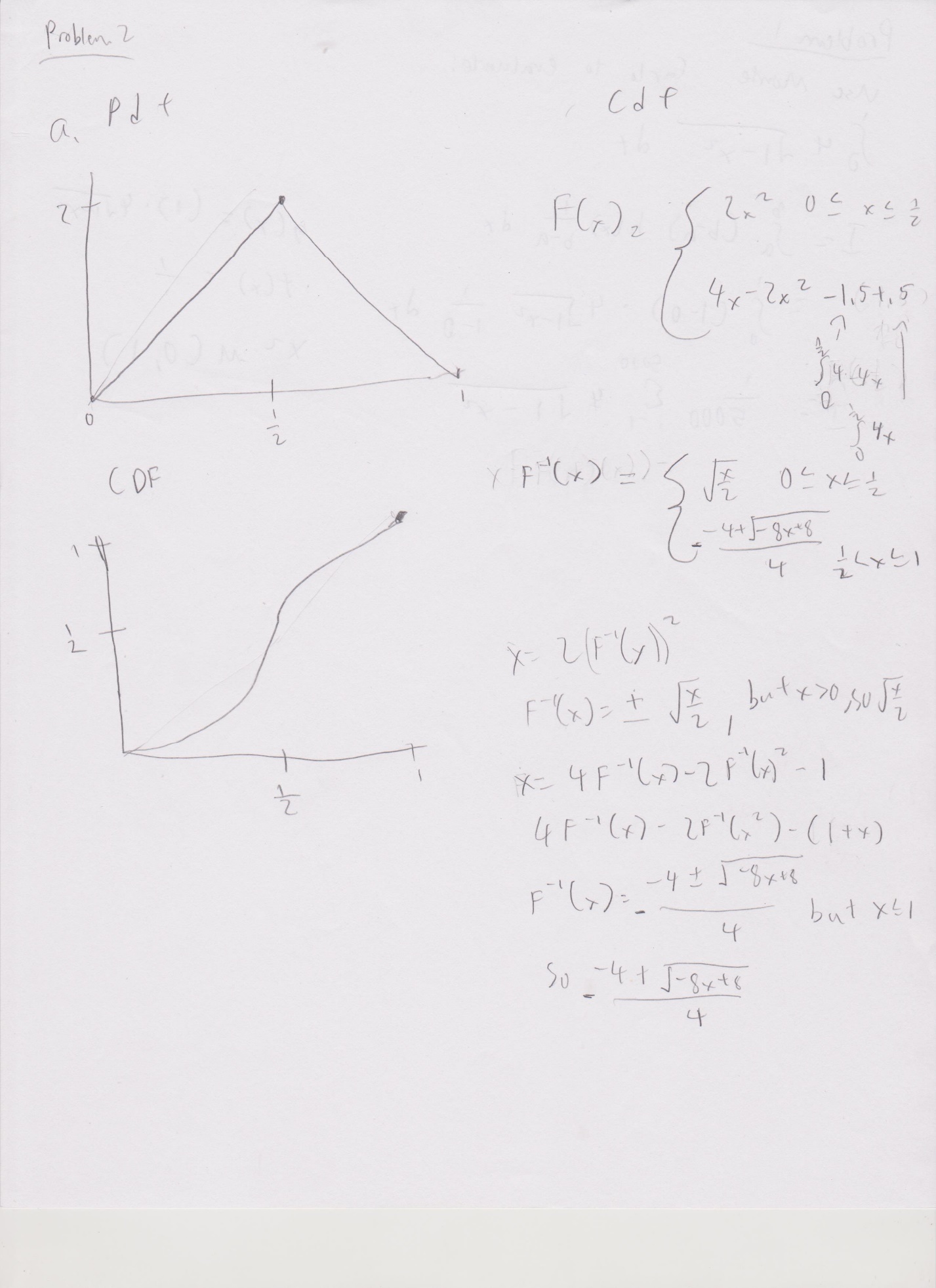
## [1] 3.140562

If we solved this analytically, we would get pi, and our approximation is very close to the real answer.



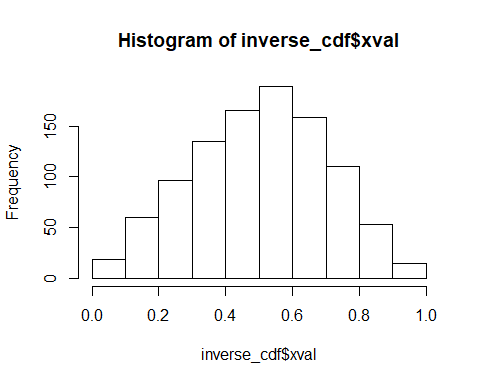
## Problem 2

### Inverse CDF



Below we will try to use the Inverse CDF to generate 1000 data points from f(x). We can see that the distribution looks very similar to the triangle pdf that we would expect.

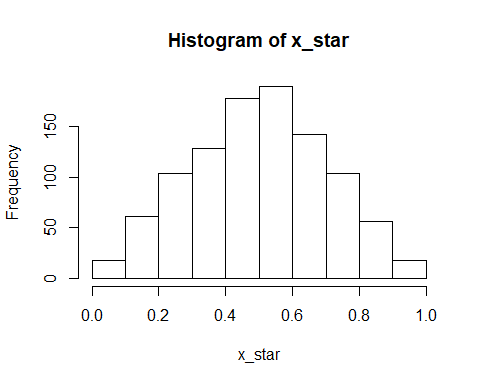
set.seed(121)  
x\_1 = runif(1000,0,1)  
inverse\_cdf = data.frame(x\_1)  
inverse\_cdf = inverse\_cdf %>%  
 mutate(xval = ifelse(x\_1<=.5, sqrt(x\_1/2),-(-4+sqrt(-8\*x\_1+8))/4))  
hist(inverse\_cdf$xval, breaks = 10)



### Rejection sampling

Suppose we take a g(x) to be a uniform distribution at y=1 from 0 to 1. We can easily see that the cdf from 0 to 1 is 1. In order for this to encompass our f(x), we need to divide by alpha = .5, so that y=2, which is that max of our original f(x). This will be our e(x).

x\_star = c()  
seed = 1  
while(length(x\_star)<1000){  
 set.seed(seed)  
 x\_s= runif(1,0,1)  
 u = runif(1,0,1)  
 f\_x\_s = ifelse(x\_s<=.5,4\*x\_s,4-4\*x\_s)  
 if(u<f\_x\_s/2){  
 x\_star = c(x\_star,x\_s)  
 }  
 seed = seed + 1  
}  
hist(x\_star)

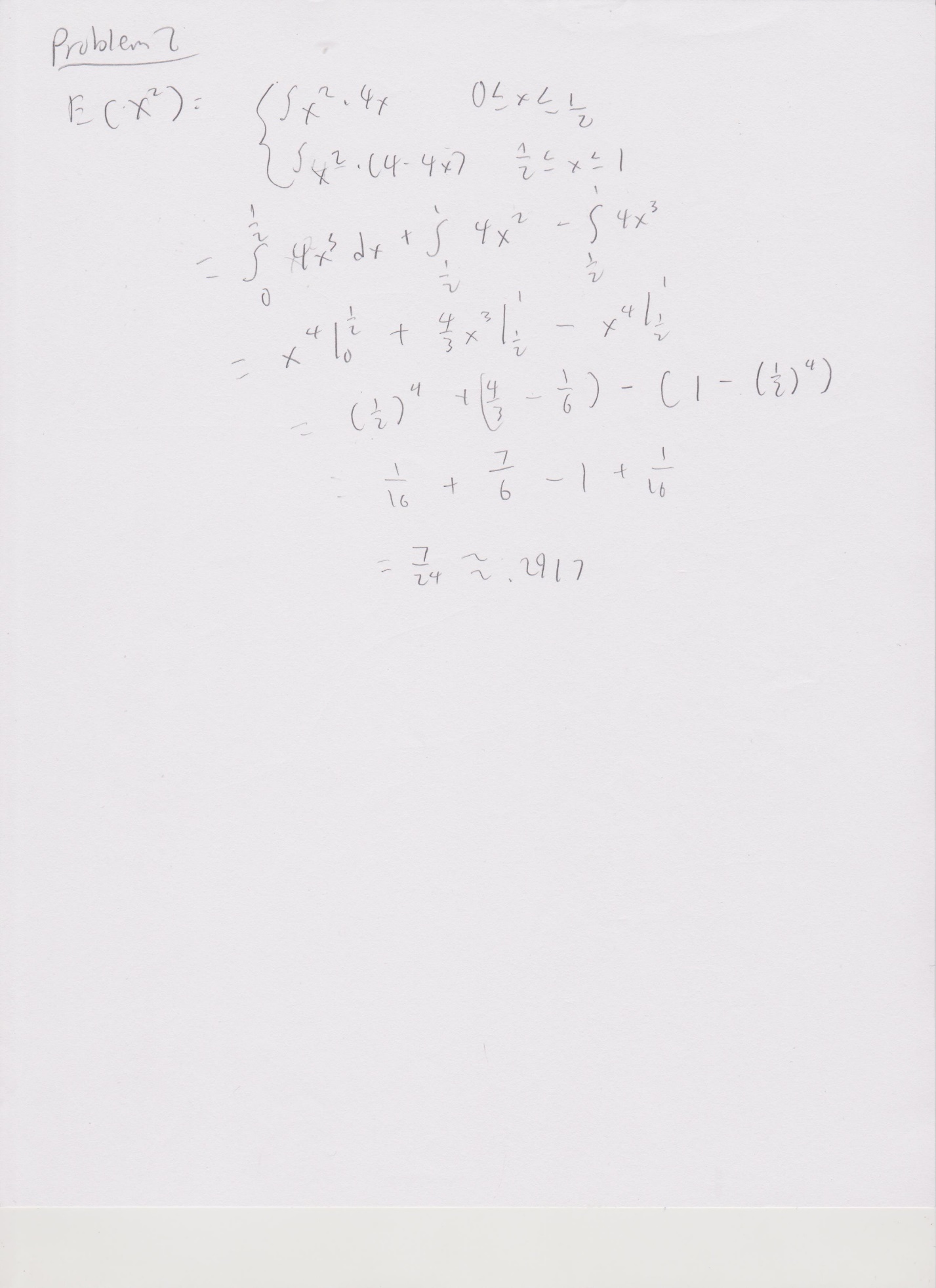


### Monte Carlo for E(X^2)

Here I used the values from the inverse CDF method. The value I got analytically was .2917, which is very close to the answer we get here.

sum(inverse\_cdf$xval^2)/length(inverse\_cdf$xval)

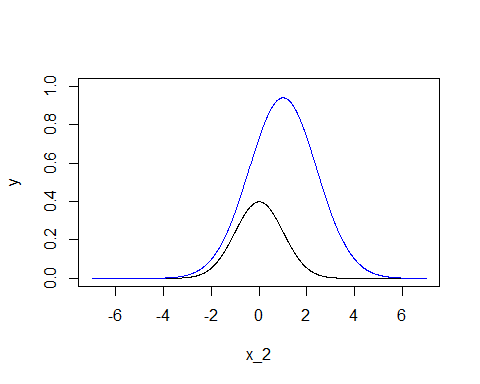
## [1] 0.2954046



## Problem 3

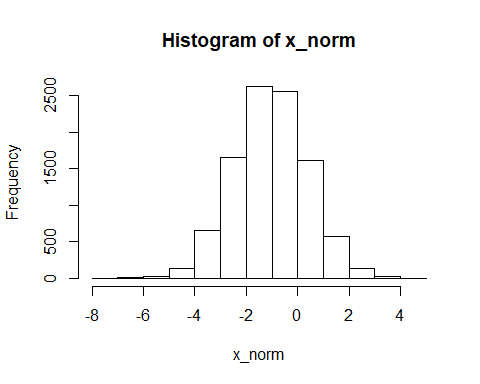
Here we try to see what alpha value would give us an envelope that could encapture all of f(x). We see that we might be violating one of the rules of rejection sampling, which is that the envelope does not result in too many rejections. We can see that an alpha of .3 applied to the N(1,2) curve would give us an envelope at least around -5 to 5. However, we can see that the enveloped is very skewed, resulting in a lot of rejections at x>0 and much fewer at x<0.

x\_2 <- seq(-7, 7, length=1000)  
y = 1/sqrt(2\*pi\*1)\*exp(-(x\_2-0)^2/(2\*1))  
z = 1/sqrt(2\*pi\*2)\*exp(-(x\_2-1)^2/(2\*2))/.3  
plot(x\_2, y, type="l", lwd=1,ylim = c(0,1))  
lines(x\_2,z, col= "blue")



Here we will run our rejection sampling and calculate our statistics.

seed = 50  
x\_norm = c()  
while(length(x\_norm) < 10000){  
 set.seed(seed)  
 x\_n = runif(1,-10,10)  
 u = runif(1,0,1)  
 f\_x\_n = 1/sqrt(2\*pi\*1)\*exp(-(x\_n-0)^2/(2\*1))  
 g\_x\_n = 1/sqrt(2\*pi\*2)\*exp(-(x\_n-1)^2/(2\*2))/.3  
 if(u<=f\_x\_n/g\_x\_n){  
 x\_norm = c(x\_norm,x\_n)  
 }  
 seed = seed + 1  
}  
hist(x\_norm)



mean(x\_norm)

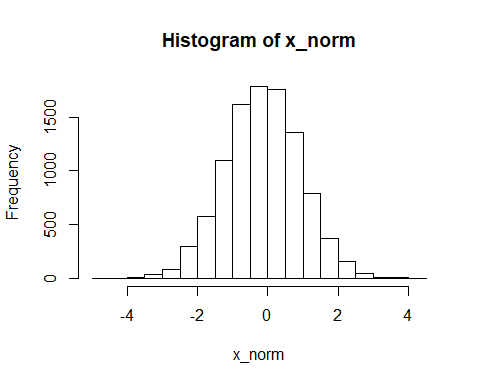
## [1] -1.02746

var(x\_norm)

## [1] 1.996584

Wait, that doesn’t seem right. What could be happening here? My theory is that the bias of the envelope is causing a lot more rejections than it should in the right side of the graph. Indeed, if we change just the variance of g\_x to 9, we see a much better result:

seed = 50  
x\_norm = c()  
while(length(x\_norm) < 10000){  
 set.seed(seed)  
 x\_n = runif(1,-10,10)  
 u = runif(1,0,1)  
 f\_x\_n = 1/sqrt(2\*pi\*1)\*exp(-(x\_n-0)^2/(2\*1))  
 g\_x\_n = 1/sqrt(2\*pi\*9)\*exp(-(x\_n-1)^2/(2\*9))/.3  
 if(u<=f\_x\_n/g\_x\_n){  
 x\_norm = c(x\_norm,x\_n)  
 }  
 seed = seed + 1  
}  
hist(x\_norm)



mean(x\_norm)

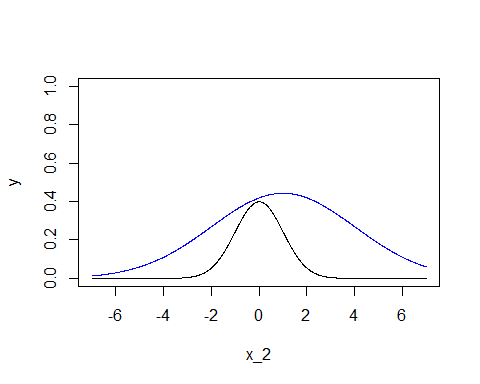
## [1] -0.1437951

var(x\_norm)

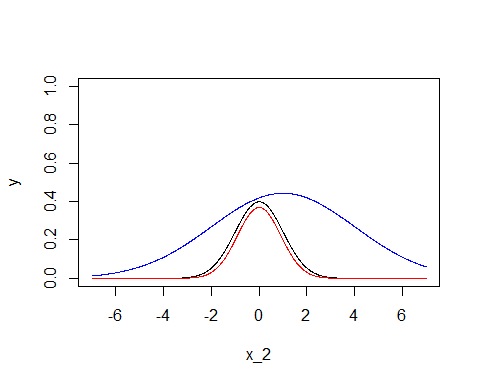
## [1] 1.132781

This is because the envelope has a much less chance of rejecting in just one region.

x\_2 <- seq(-7, 7, length=1000)  
y = 1/sqrt(2\*pi\*1)\*exp(-(x\_2-0)^2/(2\*1))  
z = 1/sqrt(2\*pi\*9)\*exp(-(x\_2-1)^2/(2\*9))/.3  
sq = 1/sqrt(2\*pi\*.81)\*exp(-(x\_2-0)^2/(2\*.81))/1.2  
plot(x\_2, y, type="l", lwd=1,ylim = c(0,1))  
lines(x\_2,z, col= "blue")

 ### Squeezed rejection The advantage to using this kind of function is that it would simplify calculations by a lot, especially if f(x) was hard to calculate. In this case, if a function falls under the squeezing function, we know that it also falls under f(x) and no additional calculation is needed for f(x). A squeezing function that lies close to f(x), but is still below it, would mean that a large number of values would have a simplified calculation.

x\_2 <- seq(-7, 7, length=1000)  
y = 1/sqrt(2\*pi\*1)\*exp(-(x\_2-0)^2/(2\*1))  
z = 1/sqrt(2\*pi\*9)\*exp(-(x\_2-1)^2/(2\*9))/.3  
sq = 1/sqrt(2\*pi\*.81)\*exp(-(x\_2-0)^2/(2\*.81))/1.2  
plot(x\_2, y, type="l", lwd=1,ylim = c(0,1))  
lines(x\_2,z, col= "blue")  
lines(x\_2,sq,col = "red")

 So one option for a squeezing function is to have another normal function with a smaller variance, with a alpha>1 applied to it so that it is always below f(x). Of course, the amount of computational time saved in this example is probably not much.