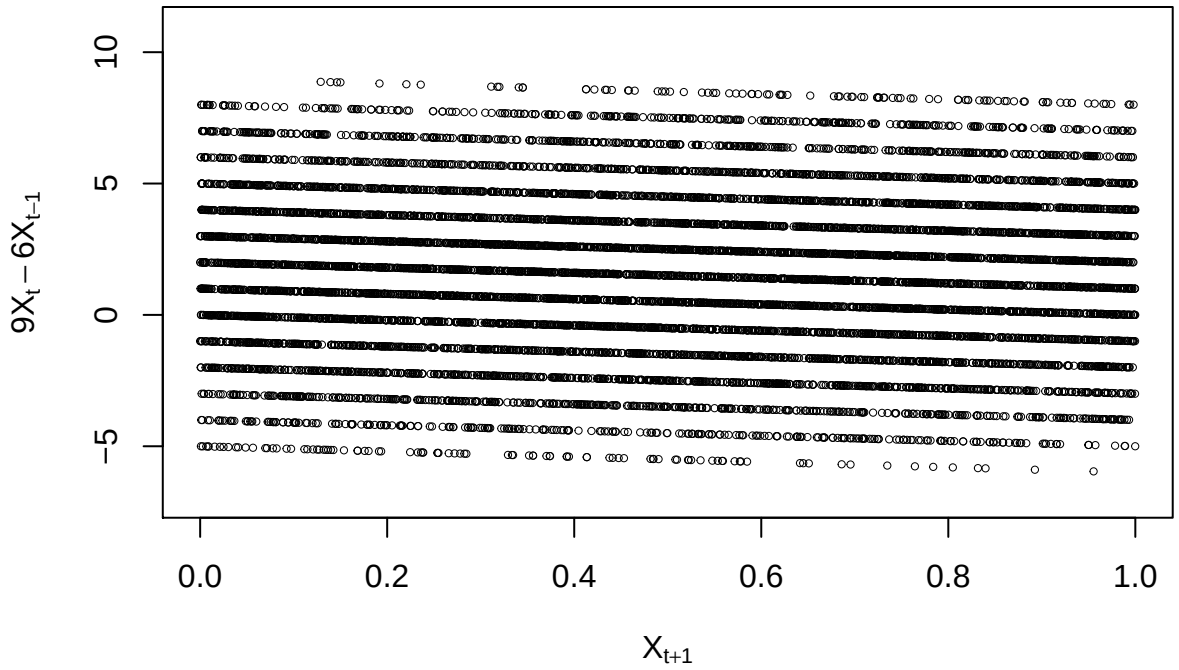
HW01

Zach White

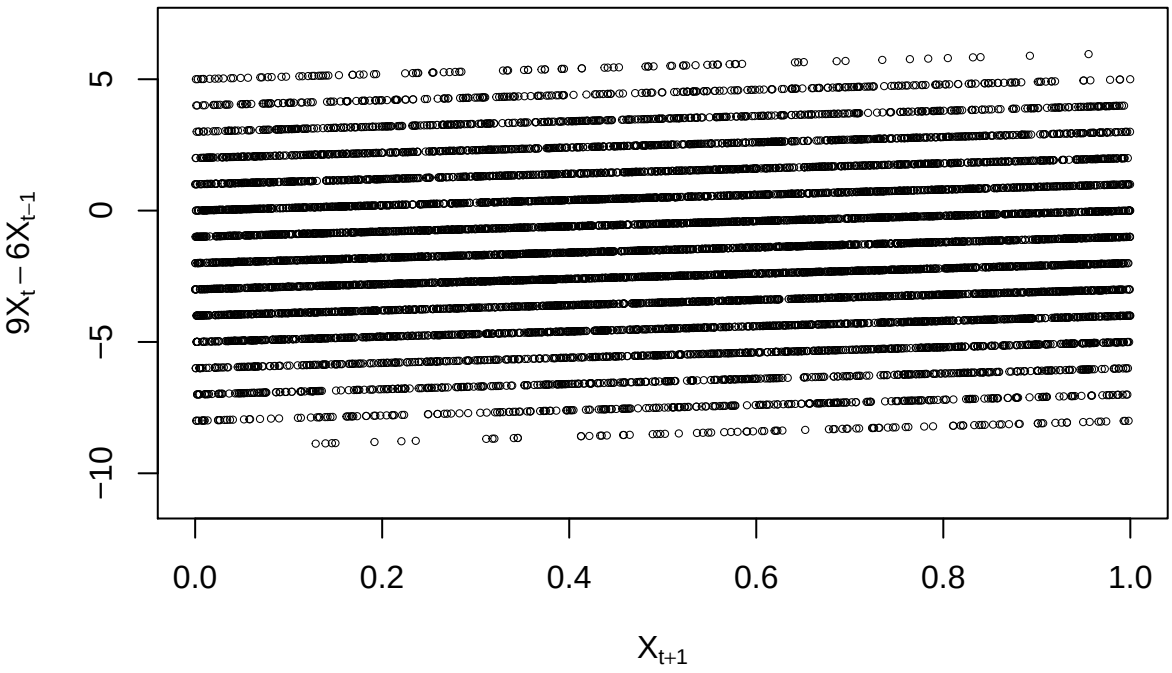
1/30/2017

Problem 1

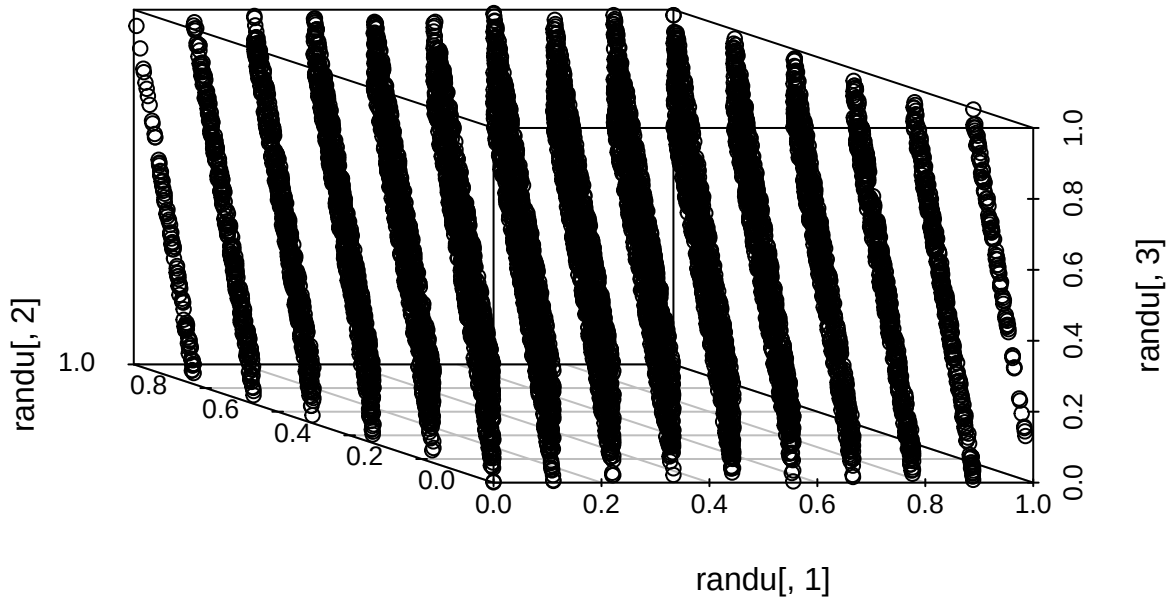
```
N = 10000
randu = matrix(0,ncol = 3, nrow = N)
x3 =1
for(i in 1:N){
  x1 = (65539 * x3) %% 2^31
  x2 = (65539 * x1) %% 2^31
  x3 = (65539 * x2) %% 2^31
  randu[i,] = c(x = x1/ 2^31,y = x2/ 2^31,z = x3/ 2^31)
}
plot(randu[,3], (9*randu[,1] - 6*randu[,2]), ylim = c(-7,11), pch = 21, lwd = .5, xlab = expression(X[t-
     ylab = expression(9*X[t] - 6*X[t-1]), cex=0.5)
```




```
plot(randu[,3], (-9*randu[,1] + 6*randu[,2]), ylim = c(-11,7), pch = 21, lwd = .5, xlab = expression(X[
     ylab = expression(9*X[t] - 6*X[t-1]), cex=0.5)
```




```
scatterplot3d(randu[,1], randu[,2], randu[,3],
angle=150)
```

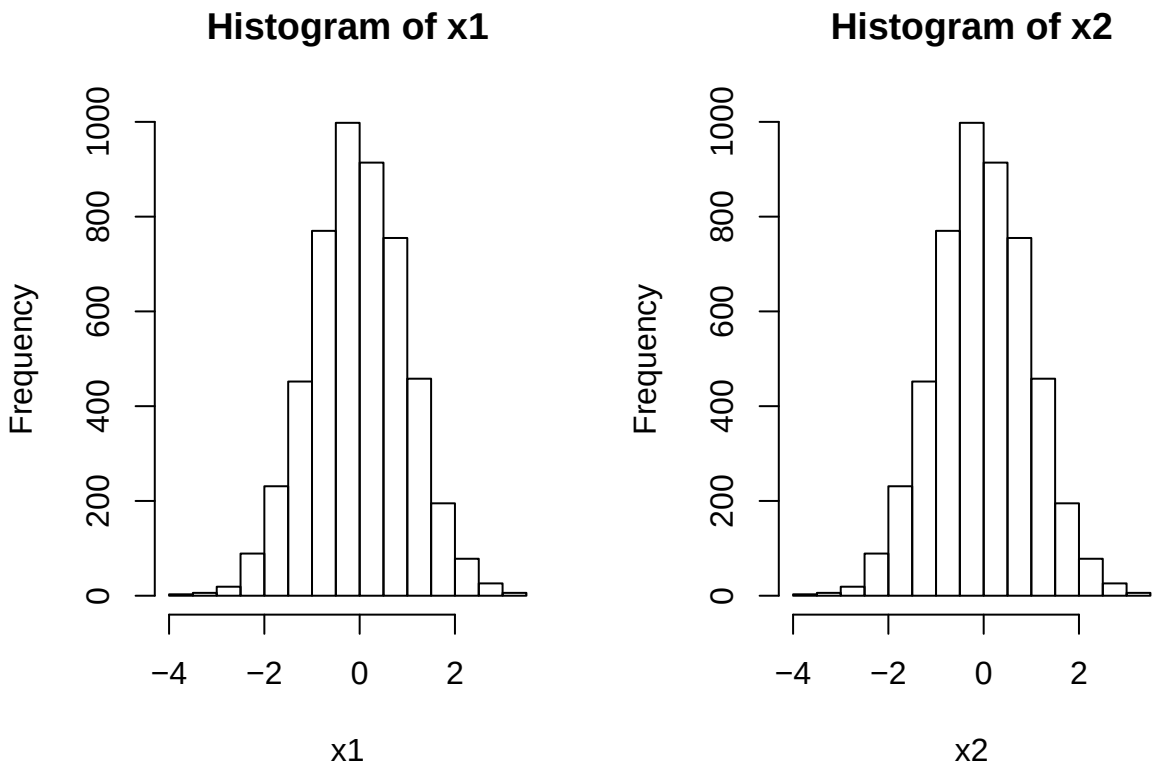


We can clearly see through this illustration the problem with this random number generator. Ther is clearly a relationship between x_{t-1}, x_t, and x_{t+1}.

Problem 2

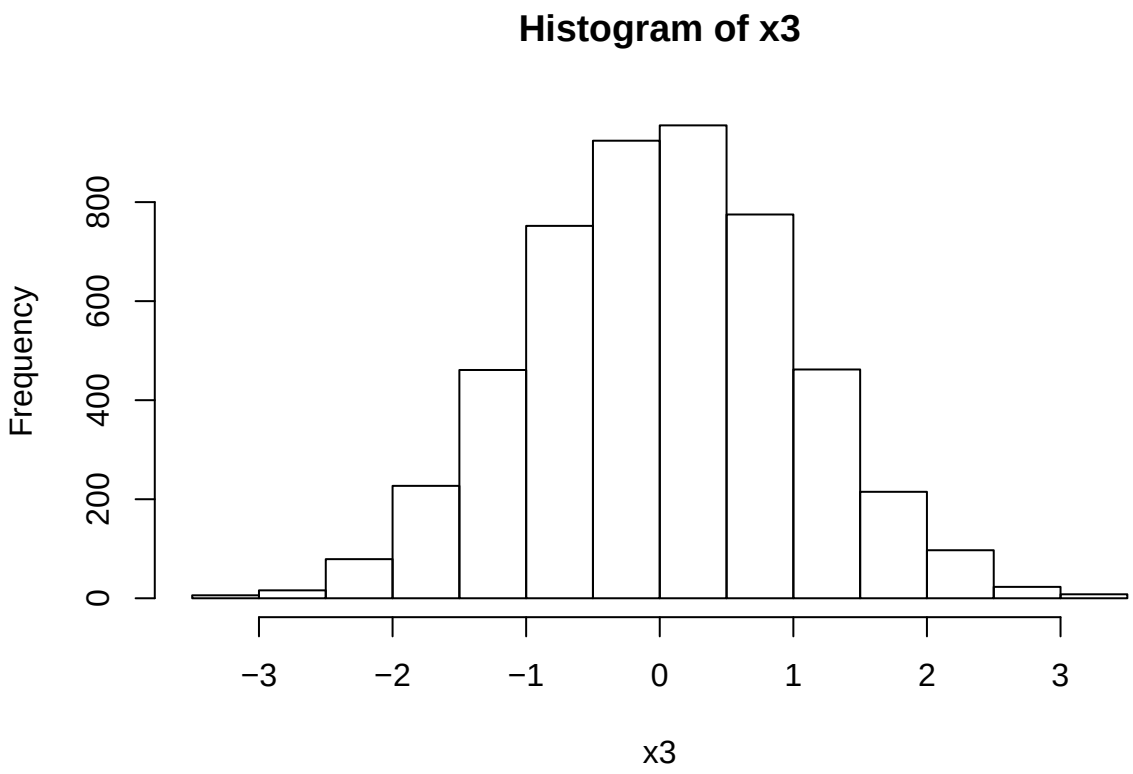
Part C

```
n = 5000
u1 = runif(n,0,1)
u2 = runif(n,0,1)
x1 = sqrt(-2*log(u1))*cos(2*pi*u2)
x2 = sqrt(-2*log(u1))*cos(2*pi*u2)
par(mfrow = c(1,2))
hist(x1)
hist(x2)
```



Part D

```
norm.approx = function(){
  u1.12 = runif(12,0,1)
  X = sum(u1.12) - 6
}
x3 = replicate(n,norm.approx())
hist(x3)
```




```
ks.test(x1,x3)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x1 and x3
## D = 0.0248, p-value = 0.09235
## alternative hypothesis: two-sided
```

```
ks.test(x2,x3)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x2 and x3
## D = 0.0248, p-value = 0.09235
## alternative hypothesis: two-sided
```

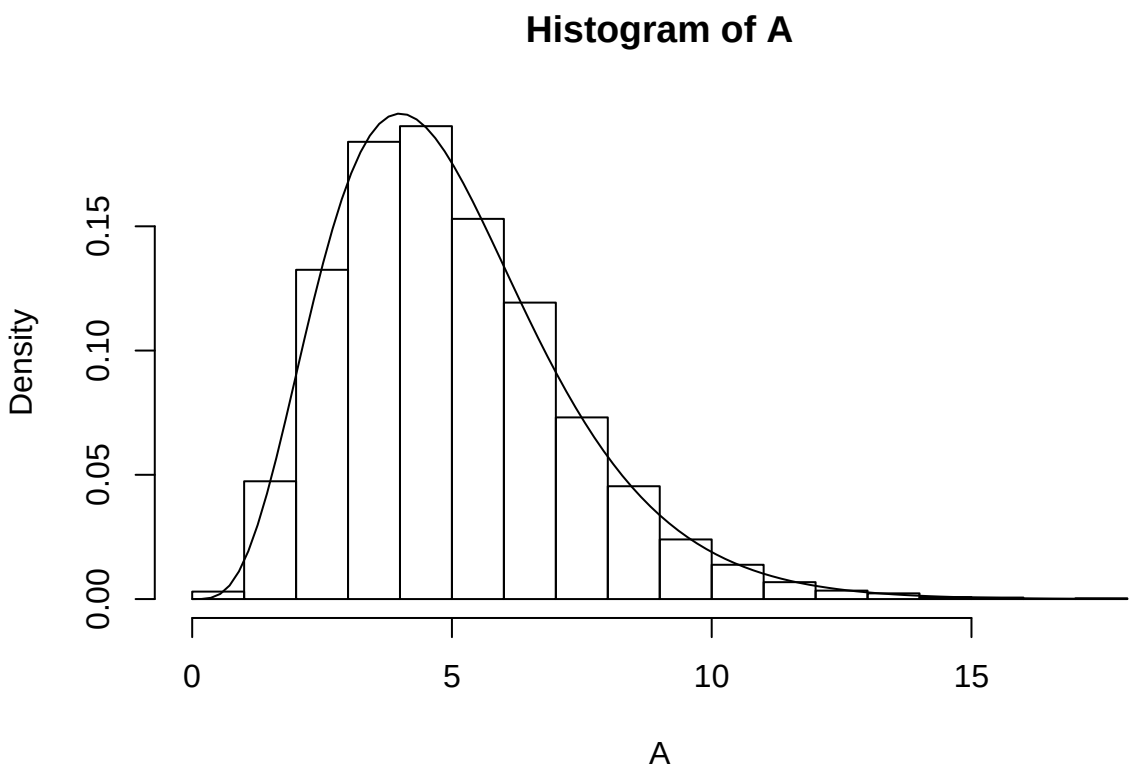
By the Kolmogorov-Smirnov test, we can say that these two sampling methods do indeed draw from the same distribution. However, I only achieved this with very high n. With smaller n, the p-value was not significant.

Problem 3

Part B

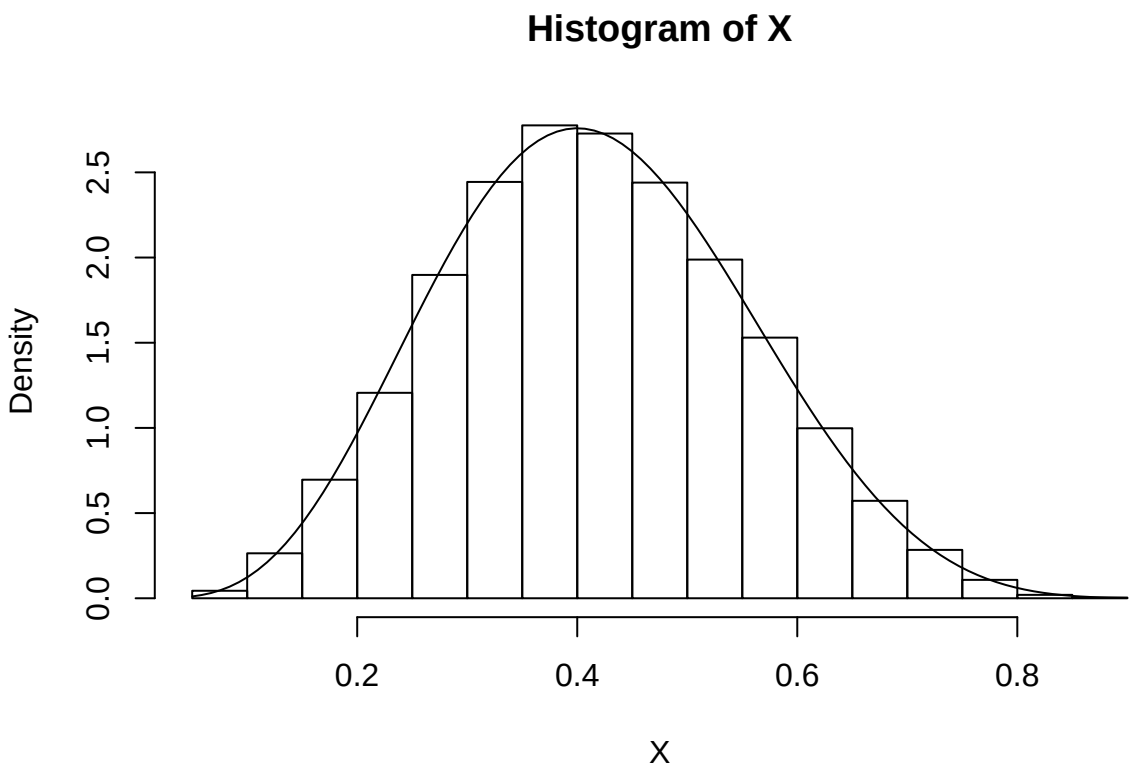
```
nreps = 10000
alpha = 5
beta = 7
```

```
gamma.approx = function(alpha){
  E1 = rexp(alpha,1)
  A = sum(E1)
}
A = replicate(nreps,gamma.approx(alpha))
hist(A, freq = FALSE)
curve(dgamma(x,alpha,1), add = TRUE)
```



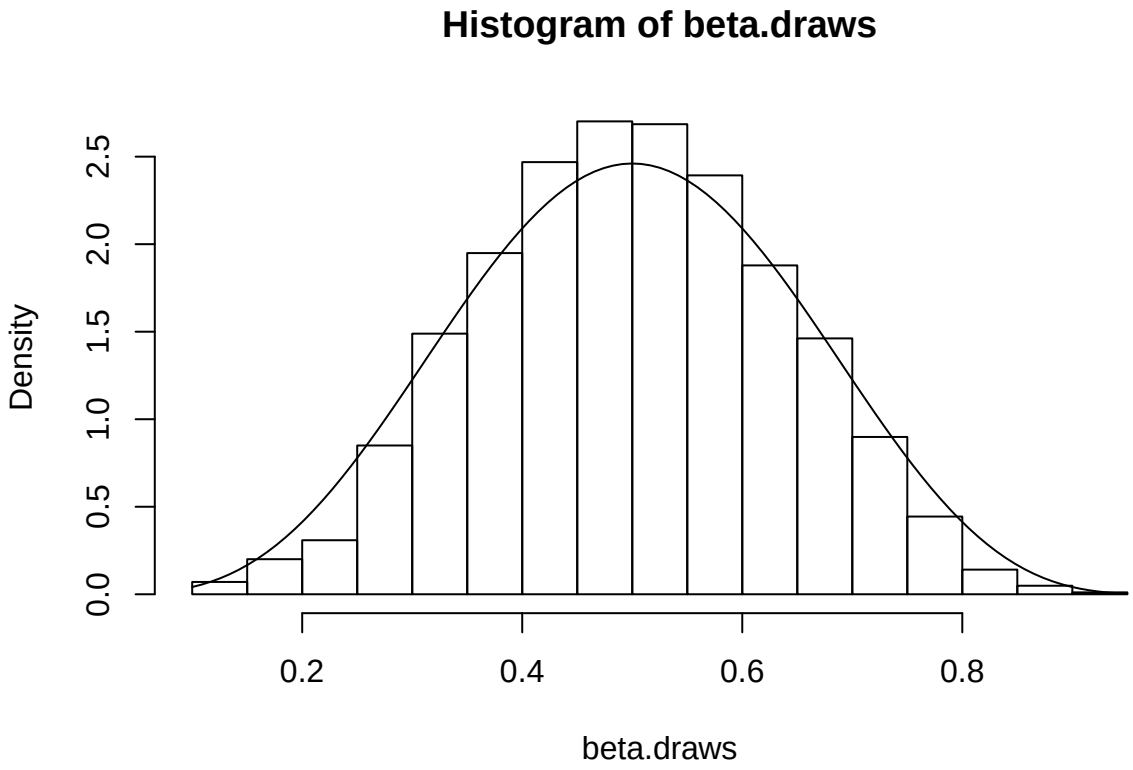

```
B = replicate(nreps,gamma.approx(beta))

X = A / (A+B)
hist(X, freq = FALSE)
curve(dbeta(x,alpha,beta), add = TRUE)
```



Part C

```
nreps = 10000
samps = rep(NA,1000)
alpha = 5
beta = 5
for(i in 1:nreps){
  u1 = runif(1,0,1)
  u2 = runif(1,0,1)
  test = 4^alpha * (u1*(1-u1))^alpha
  if(u2 <= test){samps[i] = u1}
}
beta.draws = samps[is.na(samps) == FALSE]
hist(beta.draws, freq = FALSE)
curve(dbeta(x,5,5), add = TRUE)
```

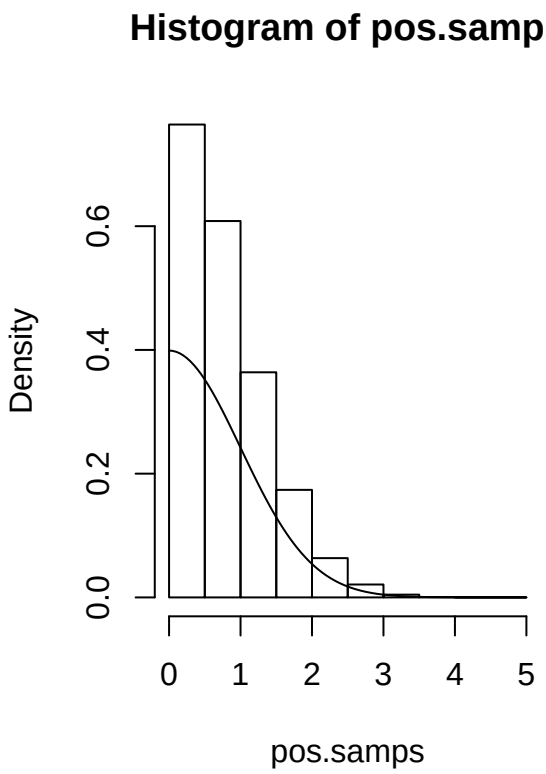


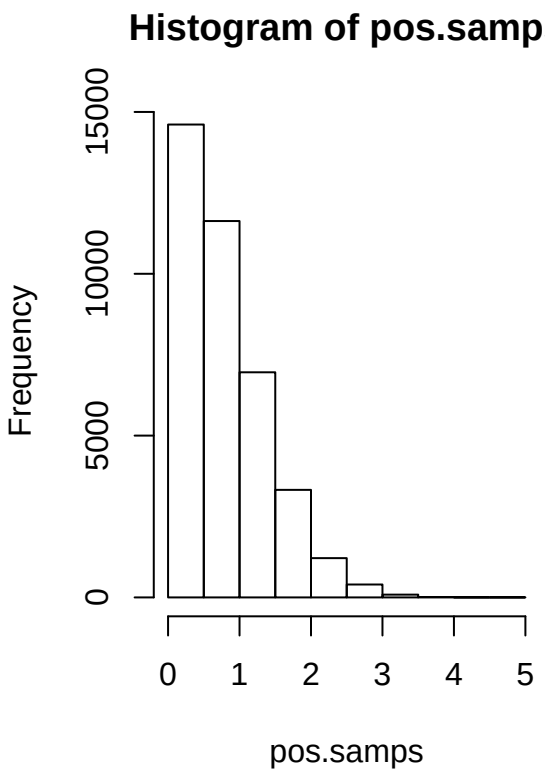
Thus this clearly draws from a beta distribution where $\alpha = \beta$. However, this code can easily be changed for cases where this equality doesn't hold.

Problem 5

```
#Shouldn't be too hard
nreps = 100000
samps = rep(NA,nreps)
cutoff1 = pnorm(1,0,1)
cutoff2 = pnorm(2,0,1)
cutoff3 = pnorm(4,0,1)

for(i in 1:nreps){
  u1 = runif(1,0,1)
  y = -log(u1)
  u2 = runif(1,0,1)
  test = exp((-(y-1)^2 )/2)
  if(u2 <= test){samps[i] = abs(y)}
  u3 = runif(1,0,1)
  if(u3 <= .5){samps[i] = -1 * samps[i]}
  #if(runif(1,0,1) <= cutoff1){}
}
real.samps = samps[is.na(samps) == FALSE]
pos.samps = real.samps[real.samps >=0]
par(mfrow = c(1,2))
hist(pos.samps, freq = FALSE)
curve(dnorm(x,0,1), add = TRUE)
hist(pos.samps)
```





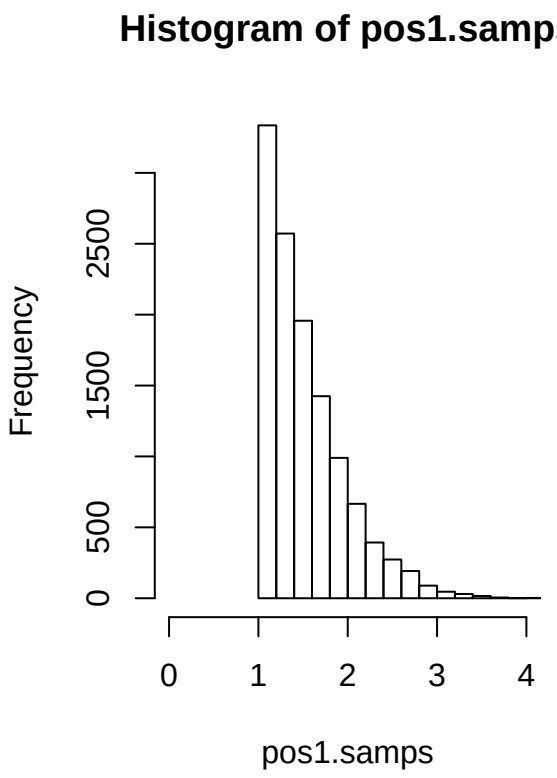

```
pos1.samps = real.samps[real.samps >= 1]
length(pos1.samps) / nreps
```

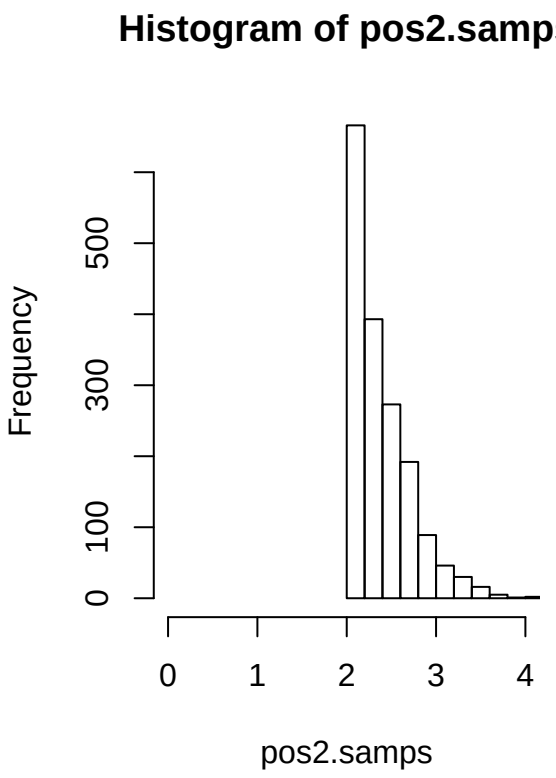
```
## [1] 0.11994
```

```
hist(pos1.samps,xlim = c(0,4))
pos2.samps = real.samps[real.samps >= 2]
length(pos2.samps) / nreps
```

```
## [1] 0.01715
```

```
hist(pos2.samps, xlim = c(0,4))
```

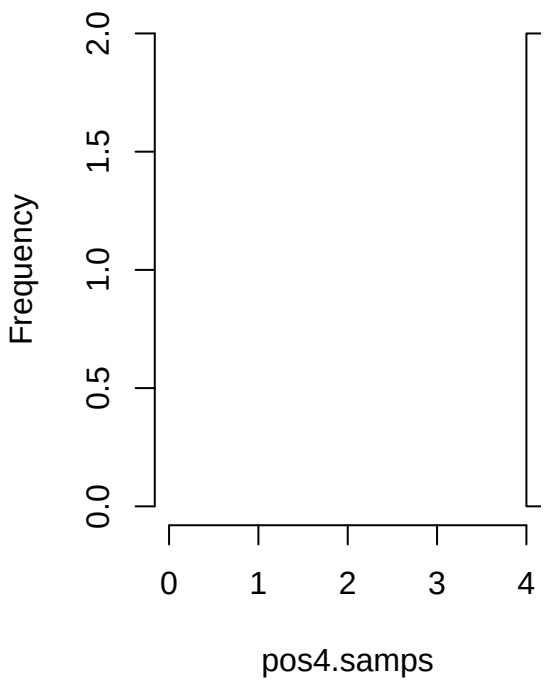





```
pos4.samps = real.samps[real.samps >= 4]
length(pos4.samps) / nreps
```

```
## [1] 4e-05
```

```
hist(pos4.samps, xlim = c(0,4))
```



The density of the histogram isn't properly weighted in this case. My method was generate from the normal distribution. These draws are from a normal distribution with a positive support, but due to my sampling scheme, they would need to be re-weighted to actually fit the curve as shown above because the curve above will integrate to 1 over $-\infty$ to ∞. Whereas, when we calculate the values of them on the positive support and make a new vector, it reweights them within that vector so the density integrates to 1.

It is clear that this method isn't super efficient, especially for lower probability areas like greater than 2 or greater than 4. This methodology could also clearly be extended to $[1, \infty), [2, \infty), [4, \infty)$ in a different way. And that was probably the intention, but I didn't do that. I would need to calculate a new M and λ for each of those cases.