Homework 3

Max Wagner

March 7, 2016

---

1. The cycle repeats 4 numbers continuously.

```
x <- 1
for(i in 1:25)
  x <- c(x, (11 * tail(x, 1)) %% 16)
x
```

```
##  [1]  1 11  9  3  1 11  9  3  1 11  9  3  1 11  9  3  1 11  9  3  1 11  9
## [24]  3  1 11
```

2.

From the 4 examples, we can see that when the when the cycle reaches 0, it immeadiatly returns to 12. When a higher number like 1000 is entered, it immeadiately decreases to 11 or lower.

```
x <- 0
for(i in 1:25)
  x <- c(x, (tail(x, 1) + 12) %% 13)
x
```

```
##  [1]  0 12 11 10  9  8  7  6  5  4  3  2  1  0 12 11 10  9  8  7  6  5  4
## [24]  3  2  1
```

```
x <- 50
for(i in 1:25)
  x <- c(x, (tail(x, 1) + 12) %% 13)
x
```

```
##  [1] 50 10  9  8  7  6  5  4  3  2  1  0 12 11 10  9  8  7  6  5  4  3  2
## [24]  1  0 12
```

```
x <- 100
for(i in 1:25)
  x <- c(x, (tail(x, 1) + 12) %% 13)
x
```

```
##  [1] 100   8   7   6   5   4   3   2   1   0  12  11  10   9   8   7   6
## [18]   5   4   3   2   1   0  12  11  10
```

```
x <- 1000
for(i in 1:25)
  x <- c(x, (tail(x, 1) + 12) %% 13)
x
```

```
##  [1] 1000   11   10    9    8    7    6    5    4    3    2    1    0   12
## [15]   11   10    9    8    7    6    5    4    3    2    1    0
```

---

3. .

```
x <- 1234567
r <- 1234567 / (2^31 - 1)

for(i in 1:99999) {
  x <- c(x, (16807 * tail(x, 1)) %% (2^31 - 1))
  r <- c(r, tail(x, 1) / (2^31 - 1))
}

tbl <- data.frame(x, r)
chisq.test(tbl)
```

```
##
##  Pearson's Chi-squared test
##
## data:  tbl
## X-squared = 1.362e-18, df = 99999, p-value = 1
```

The above is almost certainly wrong from the looks of it, not sure what to change here. Let's try runs. Trying this with a package I found.

```
library(randtests)
runs.test(r)
```

```
##
##  Runs Test
##
## data:  r
## statistic = -0.32888, runs = 49949, n1 = 50000, n2 = 50000, n =
## 100000, p-value = 0.7422
## alternative hypothesis: nonrandomness
```

4.

a. inverse-transformation

Integrating the middle function gives:

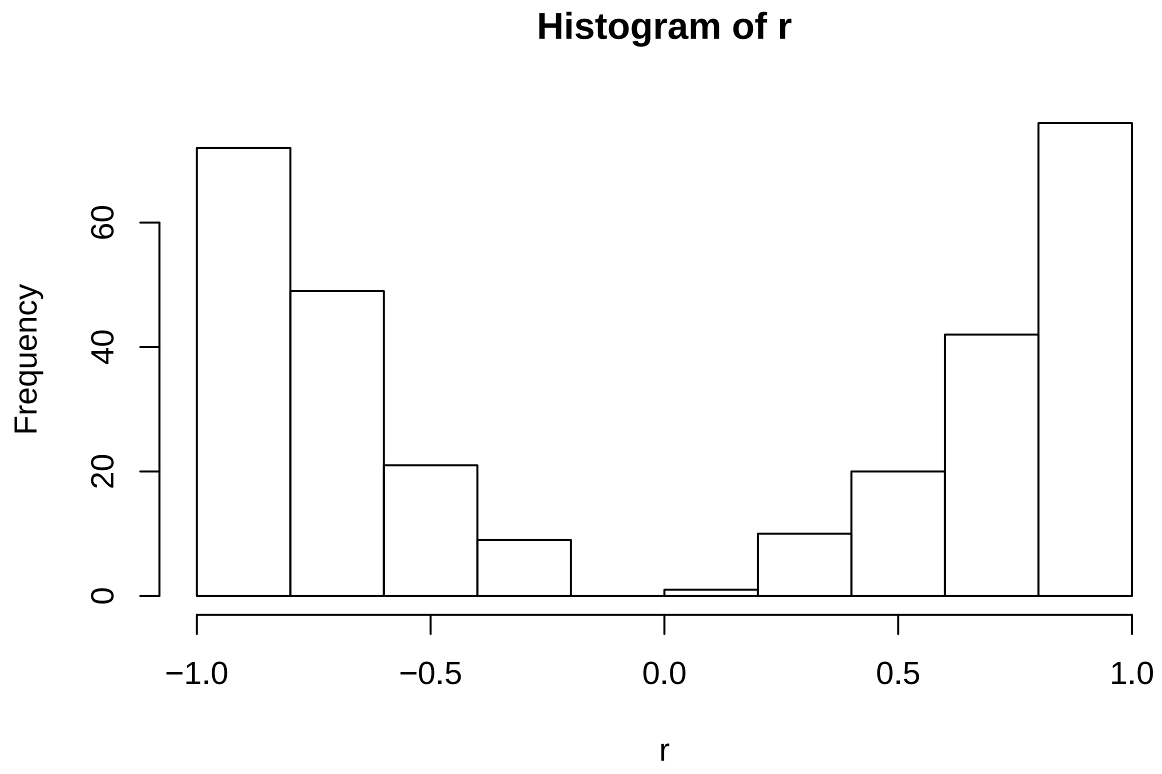
$$y = \frac{x^3 + 1}{2}$$

which then gives:

$$x = (2y - 1)^{\frac{1}{3}}, 0 \leq x \leq 1$$

```r
#cube root fxn
cbrt <- function(x) {
    return(sign(x) * abs(x)^(1/3))
}

x <- runif(300)
r <- cbrt(2*x - 1)
hist(r)
```



The above histogram looks fairly normal.

b. composition

I'll split the function over the y-axis and then copy the the positive side twice, and make one of the two generations negative. This changes the function from:

$$f(x) = \frac{3x^2}{2}$$

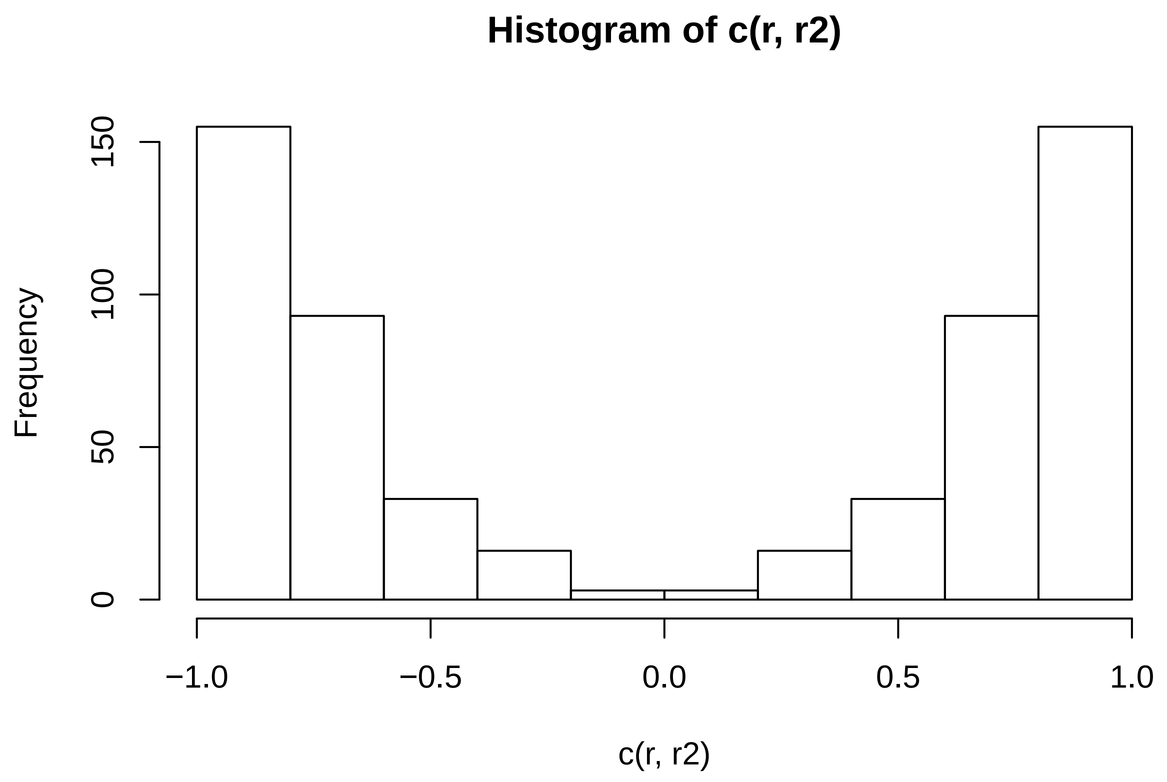
to

$$f(x) = 3x^2$$

which then makes the inverse of the CDF

$$F^{-1}(x) = (x)^{\frac{1}{3}}$$

```r
x <- runif(300)
x2 <- runif(300)
r <- x^(1/3)
r2 <- -1 * (x^(1/3))
hist(c(r,r2))
```



The histogram looks similar to that of part a, which is leads me to believe it is the intended outcome.

c. accept-reject method

First let's solve for -1 and 1 to find the edges. Plugging into

$$f(x) = \frac{3x^2}{2}$$

gives 1.5 for both -1 and 1. Not overly sure where to go from here, probably pick a random variable, but I'm not sure how to solve for the equation I would sample from. ***

5.

a. First with inverse-normal:

```r
normrandit <- function(){
  U <- runif(1)
  return(qnorm(U))
}

itstats <- function(N){
  x <- numeric(0)
  for(i in 1:N){
    x <- c(x,normrandit())
  }
  return(list(mean=mean(x), sd=sd(x)))
}
```

b. Now with Muller:

```r
normrandbm <- function(){
  U <- runif(2)
  x <- ((-2*log(U[1]))^(1/2))*cos(2*pi*U[2])
  y <- ((-2*log(U[1]))^(1/2))*sin(2*pi*U[2])
  return(c(x=x, y=y))
}

bmstats <- function(N){
  x <- numeric(0)
  for(i in 1:N){
    x <- c(x,normrandbm())
  }
  return(list(mean=mean(x), sd=sd(x)))
}
```

c. Finally with accept/reject

```r
normrandar <- function(){
  repeat{
    U <- runif(2)
    x <- -log(U[1])
    y <- -log(U[2])
    if(y >= ((x-1)^2)/2){
      break
    }
  }
  if (runif(1) >= .5)
    x <- -1 * x
  return(x)
}
```

```
arstats <- function(N){
  x <- numeric()
  for(i in 1:N){
    x <- c(x,normrandar())
  }
  return(list(mean=mean(x), sd=sd(x)))
}
```

d. Let's check the means, sd's, and runtimes

```
library(plyr)
means <- data.frame(it=c(), bm=c(), ar=c())
sds <- data.frame(it=c(), bm=c(), ar=c())
times <- data.frame(it=c(), bm=c(), ar=c())

for (n in c(100, 1000, 10000, 100000)) {
  for (i in 1:10) {
  it <- itstats(n)
  bm <- bmstats(n)
  ar <- arstats(n)
  means <- rbind(means, c(it$mean, bm$mean, ar$mean, n))
  sds <- rbind(sds, c(it$sd, bm$sd, ar$sd, n))
  times <- rbind(times, c(system.time(itstats(n))[[3]], system.time(itstats(n))[[3]], system.time(itsta
  }
}

colnames(means) <- c("it","bm","ar","n")
colnames(sds) <- c("it","bm","ar","n")
colnames(times) <- c("it","bm","ar","n")
```

```
library(plyr)
means <- read.csv("https://raw.githubusercontent.com/maxwagner/604/master/hw/means.csv")
sds <- read.csv("https://raw.githubusercontent.com/maxwagner/604/master/hw/sds.csv")
times <- read.csv("https://raw.githubusercontent.com/maxwagner/604/master/hw/times.csv")
mean_avgs <- ddply(means, ~n, summarise, mean_it = mean(it), mean_bm = mean(bm), mean_ar = mean(ar));mea
```

```
##       n       mean_it       mean_bm       mean_ar
## 1 1e+02 -0.0289959302  0.0209949281 -0.0196609236
## 2 1e+03 -0.0169933742 -0.0094790194  0.0059108701
## 3 1e+04  0.0007567270 -0.0009539500 -0.0033564852
## 4 1e+05  0.0005063963 -0.0001194233 -0.0005487528
```

```
sd_avgs <- ddply(sds, ~n, summarise, mean_it = mean(it), mean_bm = mean(bm), mean_ar = mean(ar));sd_avg
```

```
##       n   mean_it   mean_bm   mean_ar
## 1 1e+02 1.0158512 1.0057870 1.0240378
## 2 1e+03 0.9969124 0.9941471 0.9948150
## 3 1e+04 0.9987530 0.9996241 0.9981487
## 4 1e+05 0.9998424 1.0003970 1.0001452
```

```
times_avgs <- ddply(times, ~n, summarise, mean_it = mean(it), mean_bm = mean(bm), mean_ar = mean(ar));t
```

```
##       n mean_it mean_bm mean_ar
## 1 1e+02   0.001   0.001   0.000
## 2 1e+03   0.005   0.006   0.006
## 3 1e+04   0.169   0.171   0.142
## 4 1e+05  13.246  13.232  13.260
```

6.

a. and b. We don't really need to use the method of 1 or 0 to estimate pi, it should estimate fine without checking, and will not give a "4" when a lower N value is given

```
estimatepi <- function(N){
  x <- runif(N)
  y <- runif(N)
  d <- sqrt(x^2 + y^2)
  pi <- (4 * sum(d < 1.0) / N)
  se <- sd(d) / sqrt(length(d))
  ci <- qnorm(0.95)*sd(d)/sqrt(N)
  return(list(pi=pi, se=se, ci=ci))
}
```

c. Let's check how large N needs to be for the estimate to be accurate to 0.1

```
values <- data.frame(pi=c(),se=c(),ci=c())
x <- seq(1000,10000,by=500)
for (n in x) {
    values <- rbind(values, estimatepi(n))
}
rownames(values) <- seq(1000,10000,by=500)

values$lower <- 3.14 - values$ci
values$upper <- 3.14 + values$ci
values$interval <- values$upper - values$lower
values
```

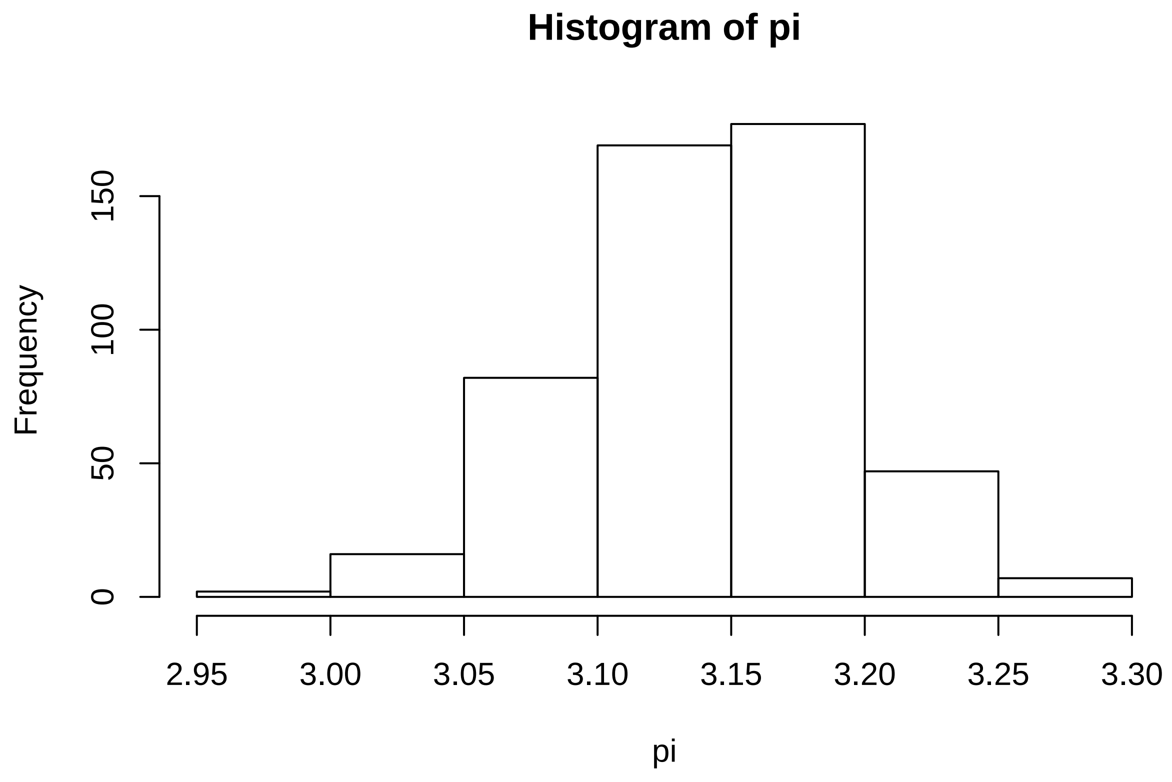
```
##             pi          se          ci    lower    upper    interval
## 1000  3.132000 0.009070032 0.014918874 3.125081 3.154919 0.029837748
## 1500  3.048000 0.007538680 0.012400025 3.127600 3.152400 0.024800050
## 2000  3.142000 0.006503671 0.010697587 3.129302 3.150698 0.021395174
## 2500  3.145600 0.005592567 0.009198954 3.130801 3.149199 0.018397908
## 3000  3.164000 0.005207707 0.008565917 3.131434 3.148566 0.017131833
## 3500  3.117714 0.004840533 0.007961968 3.132038 3.147962 0.015923936
## 4000  3.146000 0.004517894 0.007431274 3.132569 3.147431 0.014862549
## 4500  3.149333 0.004240350 0.006974754 3.133025 3.146975 0.013949509
## 5000  3.132800 0.004074694 0.006702275 3.133298 3.146702 0.013404550
## 5500  3.137455 0.003885651 0.006391328 3.133609 3.146391 0.012782656
## 6000  3.112667 0.003731569 0.006137885 3.133862 3.146138 0.012275769
## 6500  3.113846 0.003534866 0.005814337 3.134186 3.145814 0.011628674
## 7000  3.129143 0.003379024 0.005557999 3.134442 3.145558 0.011115998
## 7500  3.130667 0.003270404 0.005379335 3.134621 3.145379 0.010758671
## 8000  3.149500 0.003194182 0.005253962 3.134746 3.145254 0.010507924
## 8500  3.157176 0.003113358 0.005121018 3.134879 3.145121 0.010242036
## 9000  3.151556 0.003002617 0.004938866 3.135061 3.144939 0.009877731
## 9500  3.128421 0.002909149 0.004785125 3.135215 3.144785 0.009570250
## 10000 3.138800 0.002842981 0.004676288 3.135324 3.144676 0.009352576
```

If I believe the values above are correct, at N = 1000 the error is already small enough to be within 0.1.

d.

.

```
pi <- c()
for(i in 1:500){
  pi <- c(pi, estimatepi(1000)$pi)
}
hist(pi)
```



```
sd(pi)
```

```
## [1] 0.04972935
```

```r
count <- 0
for(i in 1:500) {
  if (pi[i] >= 3.125014 & pi[1] <= 3.154986) {
    count <- count + 1
  }
}
percentage <- count/length(pi)
```

The histogram looks normal, the sd is lower than the error from above, but that makes sense. With more estimates the error should be smaller. The percentage of estimates within the interval is 0.