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. Decreasing the 100,000 number a bit so my computer doesn't fail the calc.

```
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. .

5.

a. First with inverse-normal:

```
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:

```
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

```
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("means.csv")
sds <- read.csv("sds.csv")
times <- read.csv("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.
   - 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.

```
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  3.076000 0.008764781 0.014416782 3.125583 3.154417 0.028833564
## 1500  3.096000 0.007496967 0.012331414 3.127669 3.152331 0.024662828
## 2000  3.186000 0.006277706 0.010325908 3.129674 3.150326 0.020651816
## 2500  3.148800 0.005708940 0.009390371 3.130610 3.149390 0.018780743
## 3000  3.137333 0.005195896 0.008546488 3.131454 3.148546 0.017092977
## 3500  3.192000 0.004811151 0.007913639 3.132086 3.147914 0.015827278
## 4000  3.156000 0.004481106 0.007370764 3.132629 3.147371 0.014741528
## 4500  3.148444 0.004263198 0.007012337 3.132988 3.147012 0.014024674
## 5000  3.156800 0.004032255 0.006632469 3.133368 3.146632 0.013264939
## 5500  3.139636 0.003845109 0.006324641 3.133675 3.146325 0.012649282
## 6000  3.146000 0.003667488 0.006032482 3.133968 3.146032 0.012064963
## 6500  3.179077 0.003471730 0.005710487 3.134290 3.145710 0.011420975
## 7000  3.125143 0.003396492 0.005586731 3.134413 3.145587 0.011173463
## 7500  3.122667 0.003319845 0.005460659 3.134539 3.145461 0.010921317
## 8000  3.148500 0.003186021 0.005240539 3.134759 3.145241 0.010481078
## 8500  3.116235 0.003121408 0.005134259 3.134866 3.145134 0.010268518
## 9000  3.145333 0.002977020 0.004896762 3.135103 3.144897 0.009793524
## 9500  3.156211 0.002918253 0.004800099 3.135200 3.144800 0.009600197
## 10000 3.128800 0.002873611 0.004726670 3.135273 3.144727 0.009453340
```