HW1

Park Jeong Jin(2014150137)

2018. 4. 5

## 1

# (a)

F(x)

a <- function(x, n) {  
 1/(sqrt(2\*pi)\*factorial(n))\*((-x/2)^n)\*((n+1)/(2\*n+1))  
}  
  
  
  
  
F <- function(x) {  
 sum <- 0  
 temp <- 2  
 eps <- 0.000001  
 n<-0  
 while(abs(temp) > eps) {  
 temp <- a(x,n)\*(x^(n+1))/(n+1)  
 sum <- sum + temp  
 n <- n+1  
 }  
 return(sum)  
}

F(1)

## [1] 0.3413448

pnorm(1) - 0.5

## [1] 0.3413447

# (b)

Finding Variable y’s(>0)

and u

n <- rnorm(2000)  
y <- subset(n, n > 0)  
  
b <- length(y)  
u <- numeric(b)  
for(i in 1:b) {  
 u[i] <- F(y[i])  
}

Chi-square test of u in Uniform(0,1)

rng.chisq.test <- function(x,m) {  
 Obs.1 <- table(trunc(m\*x)/m)  
 Obs <- c(Obs.1, rep(0, m-length(Obs.1)))  
 Exv <- length(x)\*rep(1,m)/m  
 chival <- sum((Obs-Exv)^2/Exv)  
 pval <- 1-pchisq(chival,m-1)  
 list(test.stat=chival, p.value=pval, degf=m-1)}  
  
rng.chisq.test(u, 10)

## $test.stat  
## [1] 1033.01  
##   
## $p.value  
## [1] 0  
##   
## $degf  
## [1] 9

P-value is so low, which means it doesn’t follow Unif(0,1)

# (c)

Chi-square test of u in Unif(0, 1/2)

rng.chisq.test.1 <- function(x,m){  
 Obs <- table(trunc(m\*x)/m)  
 Exv <- length(x)\*rep(1,m/2)/(m/2)  
 chival <- sum((Obs-Exv)^2/Exv)  
 pval <- 1-pchisq(chival,m-1)  
 list(test.stat=chival, p.value=pval, degf=m-1)}  
  
rng.chisq.test.1(u,10)

## $test.stat  
## [1] 1.504854  
##   
## $p.value  
## [1] 0.9971106  
##   
## $degf  
## [1] 9

P-value is sufficiently high, which means it follows Unif(0, 1/2)

User-creat Varables “u.1”" which follows Unif(0, 1/2)

myrng <- function(n,a,c,m,seed){  
 x <- numeric(n)  
 x[1] <- seed  
 for (i in 1:n) {  
 x[i+1]<-(a\*x[i]+c)%%m  
 }  
 x[1:n]/m }  
  
u.1 <- myrng(1000, 171, 0, 30269, 27218)/2

And do Chi-square test again for Unif(0, 1/2)

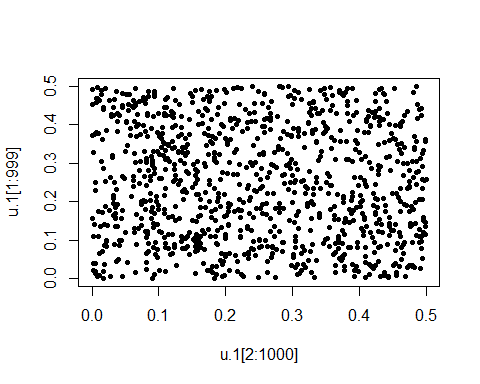
rng.chisq.test.1(u.1, 10)

## $test.stat  
## [1] 6.52  
##   
## $p.value  
## [1] 0.6869553  
##   
## $degf  
## [1] 9

it follows the distribution.

We can show their independency

plot(u.1[2:1000], u.1[1:999], pch=20)



There’s no pattern. So, we can say that they are independent.

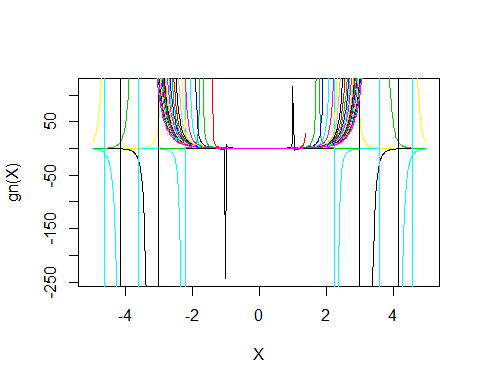
# (d)

Define the function gn(x)

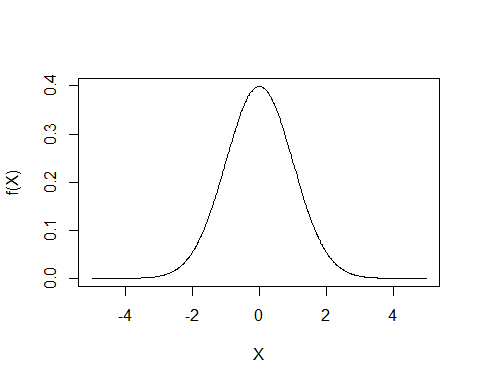
g <- function(x, n) {  
 a <- lgamma((n+1)/2)-(1/2)\*log(n\*pi)-lgamma(n/2)  
 b <- exp(a)\*(((1-((x^2)/n))^(-(n+1)/2)))  
 return(b)  
}

Draws graphs of gn(x), and f(X)

f <- function(x) {  
 (1/sqrt(2\*pi))\*exp((-(x^2)/2))  
}  
  
x <- sort(runif(1000, -5, 5))  
  
  
plot(x, g(x,1), type = "n", xlab = "X", ylab = "gn(X)")  
lines(x, g(x,1))  
for(i in 1:30) {  
 lines(x, g(x, i), col = i)  
   
}



plot(x, f(x), type = "n", xlab="X", ylab ="f(X)")  
lines(x, f(x))



gn(x) shows differences when n is even or odd number. And as n goes larger, the width of graph goes wider.

f(x) shows Standard Normal graph.

# (e)

coded the second fucntion

h <- function(n) {  
 a <-0  
 if(n == 1) return(0)  
 else {  
 a<-((1/2)\*(log(2)+log(pi)+log(n-1)-log(2)))+((n-1)/2)\*(log(n-1)-log(2)-1)  
 return(a)  
 }  
}

calculated two functions proportions.

h.1 <- numeric(1000)  
for(i in 1:1000) h.1[i] <- lgamma((i+1)/2) - h(i)

heads and tails of its result.

head(exp(h.1))

## [1] 1.000000 1.165822 1.084438 1.056344 1.042207 1.033719

tail(exp(h.1))

## [1] 1.000168 1.000168 1.000167 1.000167 1.000167 1.000167

It shows that the proportion goes close to 1 as n goes larger.

And by calculating,

h.2 <- numeric(1000)  
for(i in 1:1000) h.2[i] <- h(i)  
  
h.3 <- h.2 + log(exp(1)) - log(10)  
round(table((10^h.3) %/% (10^(floor(h.3))))/sum(table((10^h.3) %/% (10^(floor(h.3))))), 3)

##   
## 1 2 3 4 5 6 7 8 9   
## 0.379 0.164 0.124 0.102 0.051 0.051 0.045 0.045 0.040

We can see that it follows Benford’s Law

## 2

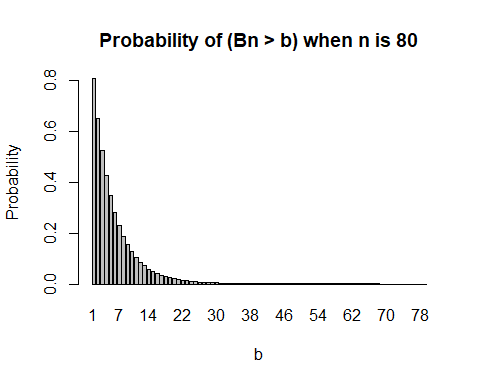
# (a)

We can calculate P(Bn > b)

#a  
B <- function(n, b) {  
 if(n != b) {  
 a <- lfactorial(365) + (n-b)\*log(365-b) - lfactorial(365-b) - n\*log(365)  
 return(exp(a))  
 }  
 else return(0)  
}

If n = 80, then we can get the graph below

Bn <- numeric(79)  
for(i in 1:79) Bn[i] <- B(80,i)  
names(Bn) <- 1:79  
  
barplot(Bn, main ="Probability of (Bn > b) when n is 80", xlab = "b", ylab = "Probability")



# (b)

We can calculate E(Bn) by,

EB <- function(n) {  
 sum <- 0  
 for(b in 1:n) {  
 temp <- b\*(B(n, b-1) - B(n, b))  
 sum <- sum + temp  
 }  
 return(sum)  
}

And we can get the graph(when n=25, 26, …, 100)

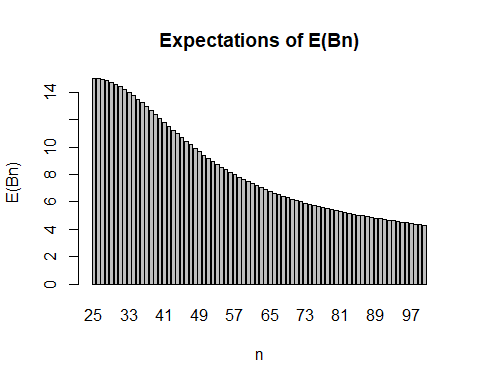
EBn <- numeric(76)  
for(i in 1:76) EBn[i] <- EB(i+24)  
head(EBn)

## [1] 15.00551 14.99232 14.94042 14.85277 14.73247 14.58263

tail(EBn)

## [1] 4.517556 4.472250 4.428035 4.384868 4.342711 4.301525

names(EBn) <- 25:100  
barplot(EBn, main ="Expectations of E(Bn)", xlab = "n", ylab = "E(Bn)")



Its E(Bn) constantly decreases.

# (c)

First, we call student one by one until there are two students who have same birthday

B.t <- function(n) {  
 a <- rep(0, 365)  
 t <- 1  
 repeat{  
 b <- sample(1:365, 1)  
 a[b] <- a[b] + 1  
 if(max(a) > 1) break  
 else if(n == t) break  
 else t <- t+1  
   
 }  
 return(t)  
}

Then, calculates sample probability with sample size 10,000.

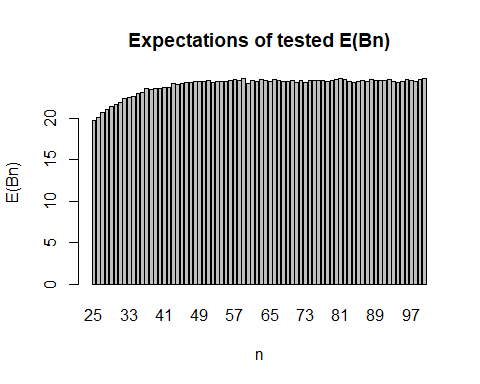
PB.t <- function(n) {  
 a <- replicate(10000, B.t(n))  
 c <- numeric(n)  
 for(i in 1:n) {  
 c[i] <- length(subset(a, a>(i)))   
 }  
 p <- c/10000  
 return(p)  
}

Lastly, calculates sample mean.

EB.t <- function(n) {  
 sum <- 0  
 p <- numeric(n+1)  
 p <- c(1, PB.t(n))  
 for(b in 1:n) {  
 temp <- b\*(p[b] - p[b+1])  
 sum <- sum + temp  
 }  
 return(sum)  
}

Then we can draw graph of sample mean.

EB.tn <- numeric(76)  
for(i in 1:76) EB.tn[i] <- EB.t(i+24)  
names(EB.tn) <- 25:100  
  
barplot(EB.tn, main = "Expectations of tested E(Bn)", xlab = "n", ylab = "E(Bn)")



It seems a little bit different from what we calculated from (b)

# 3

# Fixed Point iteration

g(x) is,

fx <- function(x, n, a) x^n - a  
fprimex <- function(x,n) n\*x^(n-1)  
  
gx <- function(x, n, a) x - fx(x, n, a)/fprimex(x, n)

And Fixed Point Iteration can be applied by,

fp <- function(ftn, x0, tol = 1e-9, max.iter = 100, n, a) {  
 n <- n  
 a <- a  
 xold <- x0  
 xnew <- ftn(xold, n, a)  
 iter <- 1  
 while ((abs(xnew-xold) > tol) && (iter < max.iter)) {  
 xold <- xnew;  
 xnew <- ftn(xold, n, a);  
 iter <- iter + 1  
 }  
 if (abs(xnew-xold) > tol) {  
 return(NULL)  
 } else {  
 return(xnew)  
 }  
}

# Newton-Raphson

We use the same f(x) and f’(x).

and Newton-Raphson’s Method can be applied by

fx <- function(x, n, a) x^n - a  
fprimex <- function(x,n) n\*x^(n-1)  
  
newt <- function(fx,fprimex,x0,epsilon, max.iter = 100, n, a){  
 diff <- 1  
 iter <- 0  
 x <- x0  
 while ((abs(diff) > epsilon) && (iter < max.iter)) {  
 if(abs(fprimex(x0, n)) < epsilon) return("incorrect specification")  
 diff <- -fx(x, n, a)/fprimex(x, n)  
 x <- x + diff  
 iter <- iter + 1  
 }  
 return(x)   
}

I will use Fixed Point iteration.

fn <- vector(length = 1000)  
for(i in 1:1000) {  
 fn[i] <- ceiling(1/(fp(gx, 1, tol = 1e-9, max.iter = 100, i, exp(1))-1))  
}  
  
head(fn)

## [1] 1 2 3 4 5 6

tail(fn)

## [1] 995 996 997 998 999 1000

We can compare this with real number.

an <- numeric(1000)  
for(i in 1:1000) an[i] <- ceiling(1/((exp(1)^(1/i))-1))  
  
head(an)

## [1] 1 2 3 4 5 6

tail(an)

## [1] 995 996 997 998 999 1000

Because the Newton Raphson’s Method use tanget line to find the answer. But this graph has no value under x=0. So, we can’t find right answer as the process goes by.