STAT323 Assignment 2

Park Jeong Jin(2014150137)

2018년 4월 17일

## 1

# (a)

Given function is,

Generating y values

y.gen <- function(x) 1 + 2\*x - 3\*x^2 + 4\*x^3 - 5\*x^4 + rnorm(1, 0, 0.01)  
y <- numeric(100)  
  
for(i in 1:100) {  
 y[i] <- y.gen(i/100)  
}  
  
head(y)

## [1] 1.028591 1.058673 1.053974 1.075765 1.089156 1.106616

tail(y)

## [1] -0.4544691 -0.5434554 -0.6479299 -0.7739896 -0.8759118 -1.0284904

## (b)

Generate matrix X and y

x.gen <- function(x) {  
 return(c(1, x, x^2, x^3, x^4))  
}  
  
X <- x.gen(1/100)  
for(i in 2:100) {  
 X<- rbind(X, x.gen(i/100))  
}  
  
beta <- matrix(c(1,2,-3,4,-5), nrow=5)  
e <- matrix(rnorm(100, 0, 0.01), ncol=1)  
  
Y <- X%\*%beta + e  
  
head(X)

## [,1] [,2] [,3] [,4] [,5]  
## X 1 0.01 0.0001 0.000001 1.000e-08  
## 1 0.02 0.0004 0.000008 1.600e-07  
## 1 0.03 0.0009 0.000027 8.100e-07  
## 1 0.04 0.0016 0.000064 2.560e-06  
## 1 0.05 0.0025 0.000125 6.250e-06  
## 1 0.06 0.0036 0.000216 1.296e-05

head(Y)

## [,1]  
## X 1.024787  
## 1.052189  
## 1.048423  
## 1.071774  
## 1.075493  
## 1.112112

Calculates Beta.hat, then compare it with real beta.

beta.hat <- solve(t(X)%\*%X)%\*%t(X)%\*%Y  
  
a <- cbind(beta.hat, beta)  
colnames(a) <- c("estimated", "real")  
  
print(a)

## estimated real  
## [1,] 0.9962421 1  
## [2,] 2.0622885 2  
## [3,] -3.2008570 -3  
## [4,] 4.2792762 4  
## [5,] -5.1365146 -5

Use lm function to see its fittedness.

lm(y~X)

##   
## Call:  
## lm(formula = y ~ X)  
##   
## Coefficients:  
## (Intercept) X1 X2 X3 X4   
## 1.006 NA 1.961 -2.915 3.920   
## X5   
## -4.975

## (c)

A <- diag(x=1, nrow=100) - X%\*%solve(t(X)%\*%X)%\*%t(X)

Eigen Values of A power 10 can be calculated through eigen()

#install.packages(expm)  
library(expm)

## Loading required package: Matrix

##   
## Attaching package: 'expm'

## The following object is masked from 'package:Matrix':  
##   
## expm

head(eigen(A%^%10)$values)

## [1] 1+0i 1-0i 1+0i 1+0i 1+0i 1+0i

We cannot use ‘Power Method’. Because, there should be difference among eigen values. But there’s not.

A1 <- A%^%10  
  
vnorm<- function(x) {  
 sqrt(sum(x \* x))  
}  
  
x0 <- as.vector(c(1,rep(0,99)))  
diff <- 1  
eps <- 0.0001  
count <- 0  
  
while(diff > eps){  
 x1 <- A1 %\*% x0  
 lambda1 <- vnorm(x1)/vnorm(x0)  
 x1 <- x1/vnorm(x1)  
 diff <- vnorm(x1-x0)  
 x0 <- x1  
 count <- count+1  
}  
  
A2 <- A1 - lambda1\*x0%\*%t(x0)   
  
  
  
x0 <- as.vector(c(1,rep(0,99)))  
diff <- 1  
eps <- 0.0001  
count <- 0  
  
while(diff > eps){  
 x1 <- A2 %\*% x0  
 lambda2 <- vnorm(x1)/vnorm(x0)  
 x1 <- x1/vnorm(x1)  
 diff <- vnorm(x1-x0)  
 x0 <- x1  
 count <- count+1  
}  
  
lambda1 - lambda2

## [1] 3.774758e-15

## (d)

Generate the real function.

fxi.gen <- function(x) {  
 return(1000\*(0.3\*(x^2)\*((1-x)^6)+0.7\*x^6\*((1-x)^2)))  
}  
  
xi <- numeric(200)  
for(i in 1:200) {  
 xi[i] <- (i-(1/2))/200  
}  
  
fxi <- numeric(200)  
for(i in 1:200) {  
 fxi[i] <- fxi.gen(xi[i])  
}  
  
ei <- rnorm(200)  
  
yi <- fxi + ei  
  
head(yi)

## [1] 0.05088519 -1.51195371 0.20082938 0.91625572 0.10749365 -0.65393393

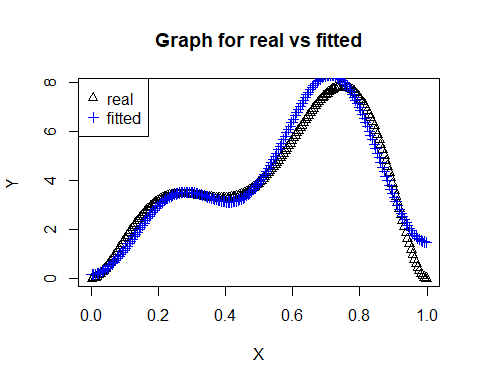
Generates the cosine regression model. Then, calculates beta.hat

cx.gen <- function(x) {  
 return(c(1, cos(pi\*x), cos(pi\*x\*2), cos(pi\*x\*3), cos(pi\*x\*4)))  
}  
  
Xi <- cx.gen(xi[1])  
for(i in 2:200) {  
 Xi <- rbind(Xi, cx.gen(xi[i]))  
}  
  
Yi <- matrix(yi, ncol=1)  
  
betai.hat <- solve(t(Xi)%\*%Xi)%\*%t(Xi)%\*%Yi  
  
print(betai.hat)

## [,1]  
## [1,] 4.042993  
## [2,] -1.951686  
## [3,] -1.543200  
## [4,] 1.294027  
## [5,] -1.683079

Compare real fx and cosine fx

fxi.hat <- Xi%\*%betai.hat  
  
plot(xi, fxi, main = "Graph for real vs fitted", xlab = "X", ylab = "Y", pch = 2)  
points(xi, fxi.hat, col = "blue", pch = 3)  
legend("topleft", c("real", "fitted"), col = c("black", "blue"), pch = c(2, 3))



# 2

## (a)

Caculates its probability and Matrix.

p1j <- c(0, 1, 0, 0, 0, 0)  
p2j <- c(1/4, 0, 1/4, 1/4, 1/4, 0)  
p3j <- c(0, 1/4, 0, 1/4, 1/4, 1/4)  
p4j <- c(0, 1/4, 1/4, 0, 1/4, 1/4)  
p5j <- c(0, 1/3, 1/3, 1/3, 0, 0)  
p6j <- c(0, 0, 1/2, 1/2, 0, 0)  
  
P <- rbind(p1j, p2j, p3j, p4j, p5j, p6j)  
colnames(P) <- 1:6  
  
print(P)

## 1 2 3 4 5 6  
## p1j 0.00 1.0000000 0.0000000 0.0000000 0.00 0.00  
## p2j 0.25 0.0000000 0.2500000 0.2500000 0.25 0.00  
## p3j 0.00 0.2500000 0.0000000 0.2500000 0.25 0.25  
## p4j 0.00 0.2500000 0.2500000 0.0000000 0.25 0.25  
## p5j 0.00 0.3333333 0.3333333 0.3333333 0.00 0.00  
## p6j 0.00 0.0000000 0.5000000 0.5000000 0.00 0.00

Then, finds the probability that the frog moves 100 times from leaf number1 to each leaves.

(P%^%100)[1,]

## 1 2 3 4 5 6   
## 0.05555556 0.22222222 0.22222222 0.22222222 0.16666667 0.11111111

## (b)

Caculates its probability and Matrix.

P1 <- matrix(rep(0,36), nrow=6)  
for(i in 1:6) {  
 for(j in 1:6) {  
 if((i-j)%%6 == 1 || (i-j)%%6 == 5) {  
 P1[i,j] <- 1/2  
 }  
 }  
}  
  
colnames(P1) <- 1:6  
  
print(P1)

## 1 2 3 4 5 6  
## [1,] 0.0 0.5 0.0 0.0 0.0 0.5  
## [2,] 0.5 0.0 0.5 0.0 0.0 0.0  
## [3,] 0.0 0.5 0.0 0.5 0.0 0.0  
## [4,] 0.0 0.0 0.5 0.0 0.5 0.0  
## [5,] 0.0 0.0 0.0 0.5 0.0 0.5  
## [6,] 0.5 0.0 0.0 0.0 0.5 0.0

Then, finds the probability that the frog moves 100 times from leaf number1 to each leaves.

(P1%^%100)[1,]

## 1 2 3 4 5 6   
## 0.3333333 0.0000000 0.3333333 0.0000000 0.3333333 0.0000000

## (c)

Do tests about question (a)

fa <- function(x0) {  
 r <- 0  
 if(x0 == 1) r <- 2   
 else if(x0 == 2) r <- sample(c(1, 3, 4, 5), 1, prob = c(1/4, 1/4, 1/4, 1/4))  
 else if(x0 == 3) r <- sample(c(2, 4, 5, 6), 1, prob = c(1/4, 1/4, 1/4, 1/4))  
 else if(x0 == 4) r <- sample(c(2, 3, 5, 6), 1, prob = c(1/4, 1/4, 1/4, 1/4))  
 else if(x0 == 5) r <- sample(c(2, 3, 4), 1, prob = c(1/3, 1/3, 1/3))  
 else if(x0 == 6) r <- sample(c(3, 4), 1, prob = c(1/2, 1/2))  
 return(r)  
}  
  
Fa <- function(x0, n) {  
 c <- 1  
 a <- fa(x0)  
 repeat{  
 a <- fa(a)  
 c <- c+1  
 if(c > n-1) break  
 }  
 return(a)  
}  
  
re <- table(replicate(1000, Fa(1, 100)))/1000  
  
  
for(i in 2:6) {  
 re <- rbind(re, table(replicate(1000, Fa(i, 100)))/1000)  
}  
  
re[1,]

## 1 2 3 4 5 6   
## 0.057 0.221 0.221 0.232 0.173 0.096

As the same, do tests about question (b)

fb <- function(x0) {  
 r <- 0  
 if(x0 == 1) r <- sample(c(2, 6), 1, prob = c(1/2, 1/2))  
 else if(x0 == 2) r <- sample(c(1, 3), 1, prob = c(1/2, 1/2))  
 else if(x0 == 3) r <- sample(c(2, 4), 1, prob = c(1/2, 1/2))  
 else if(x0 == 4) r <- sample(c(3, 5), 1, prob = c(1/2, 1/2))  
 else if(x0 == 5) r <- sample(c(4, 6), 1, prob = c(1/2, 1/2))  
 else if(x0 == 6) r <- sample(c(1, 5), 1, prob = c(1/2, 1/2))  
 return(r)  
}  
  
  
Fb <- function(x0, n) {  
 c <- 1  
 a <- fb(x0)  
 repeat{  
 a <- fb(a)  
 c <- c+1  
 if(c > n-1) break  
 }  
 return(a)  
}  
  
re1 <- matrix(rep(0, 36), nrow=6)  
  
for(i in 1:6) {  
 temp <- table(replicate(1000, Fb(i, 100)))/1000  
 re1[i, as.numeric(names(temp))] <- temp[rank(names(temp))]  
}  
  
re1[1,]

## [1] 0.308 0.000 0.339 0.000 0.353 0.000

We can see that they have similar probabilities with calculated probabilities.

# 3

## (a)

Generates its matrix.

t1j <- c(0, 8/36, 4/36, 3/36, 4/36, 5/36, 5/36, 4/36, 3/36)  
t2j <- c(0, 1, 0, 0, 0, 0, 0, 0, 0)  
t3j <- c(0, 0, 1, 0, 0, 0, 0, 0, 0)  
  
T <- matrix(c(t1j, t2j, t3j, rep(0, 6\*9)), byrow=T, nrow=9)  
  
for(i in 4:9) {  
 T[i, 2] <- T[1, i]  
 T[i, 3] <- 6/36  
 T[i, i] <- 1-T[i,2]-T[i,3]  
}  
colnames(T) <- c("S", "W", "L", "4", "5", "6", "8", "9", "10")  
rownames(T) <- c("S", "W", "L", "4", "5", "6", "8", "9", "10")  
  
print(T)

## S W L 4 5 6 8  
## S 0 0.22222222 0.1111111 0.08333333 0.1111111 0.1388889 0.1388889  
## W 0 1.00000000 0.0000000 0.00000000 0.0000000 0.0000000 0.0000000  
## L 0 0.00000000 1.0000000 0.00000000 0.0000000 0.0000000 0.0000000  
## 4 0 0.08333333 0.1666667 0.75000000 0.0000000 0.0000000 0.0000000  
## 5 0 0.11111111 0.1666667 0.00000000 0.7222222 0.0000000 0.0000000  
## 6 0 0.13888889 0.1666667 0.00000000 0.0000000 0.6944444 0.0000000  
## 8 0 0.13888889 0.1666667 0.00000000 0.0000000 0.0000000 0.6944444  
## 9 0 0.11111111 0.1666667 0.00000000 0.0000000 0.0000000 0.0000000  
## 10 0 0.08333333 0.1666667 0.00000000 0.0000000 0.0000000 0.0000000  
## 9 10  
## S 0.1111111 0.08333333  
## W 0.0000000 0.00000000  
## L 0.0000000 0.00000000  
## 4 0.0000000 0.00000000  
## 5 0.0000000 0.00000000  
## 6 0.0000000 0.00000000  
## 8 0.0000000 0.00000000  
## 9 0.7222222 0.00000000  
## 10 0.0000000 0.75000000

Show its winning probability is 0.493

W <- T%^%100  
W[1,2]

## [1] 0.4929293

## (b)

Do tests about CRAPS game.

for(i in 1:10000) {  
a[i]<-0  
d1 <- sample(1:6, 1)  
d2 <- sample(1:6, 1)  
fds <- d1+d2  
if(fds == 7 || fds == 11) a[i] <- 1  
else if(fds == 2 || fds == 3 || fds == 12) a[i] <- 0  
else repeat{  
 d1 <- sample(1:6, 1)  
 d2 <- sample(1:6, 1)  
 ds <- d1 + d2  
 if(ds == fds) {a[i]<- 1; break}  
 else if(ds == 7) {a[i]<- 0; break}  
}  
}  
  
sum(a)/10000

## [1] 0.4938

# 4

Given f(x) is,

Creats f(x) and its derivatives. And D, R.

fx <- function(x) log(x) - exp(-x)  
fxp <- function(x) 1/x + exp(-x)  
fx2p <- function(x) -1/x^2 - exp(-x)  
fx3p <- function(x) 2\*(1/x^3) + exp(-x)  
fx4p <- function(x) -6\*(1/x^4) - exp(-x)  
fx5p <- function(x) 24\*(1/x^5) + exp(-x)  
D <- function(x, h) fxp(x) + h^2/factorial(3)\*fx3p(x) + h^4/factorial(5)\*fx5p(x)  
R <- function(x, h = .Machine$double.eps, p = 2) (2^p\*D(x,h) - D(x, 2\*h))/(2^p-1)

Caculate its answer by applying Richardson extraplation to Secant method.

But first, we need to find correct ‘h’ to make x1 = 2

I used secant method

#Secant method  
secant <- function(ftn, x0, x1, tol = 1e-9, max.iter = 100){  
 f0 <- ftn(x0); f1 <- ftn(x1); iter <- 0  
 while ((abs(f1) > tol) && (iter < max.iter)) {  
 if (f0 == f1) {  
 return("Algorithm failed with f0 == f1")}  
 x2 <- x1 - f1\*(x1 - x0)/(f1 - f0)  
 x0 <- x1; f0 <- f1; x1 <- x2  
 f1 <- ftn(x1); iter <- iter + 1 }  
 return(x1) }  
  
h <- function(h) (4/3)\*D(1, h) - (1/3)\*D(1, 2\*h) - exp(-1)  
  
secant(h,0,1)

## [1] 1.053358

R(1, 1.053358)

## [1] 0.3678783

exp(-1)

## [1] 0.3678794

1-fx(1)/R(1, 1.053358)

## [1] 2.000003

Now, our h is 1.053358.

And, finally we applied Richardson extraploation to our equation.

#Richardson extrapolation(p=2)  
## x0 = 1, x1 = 2  
  
secant.Richardson <- function(x0, h = 1.053358, tol = 1e-9, max.iter = 100) {  
 nx <-0  
 ox <- x0  
 iter <- 0  
 while((abs(fx(ox)/R(ox,h)) > tol && (iter < max.iter))) {  
 nx <- ox - fx(ox)/R(ox,h)  
 ox <- nx  
 iter <- iter +1  
 }  
 return(nx)  
}  
  
secant.Richardson(1)

## [1] 1.3098

There are differences among Newton-Raphson, Fixed Point and Secant methods.  
  
Newton-Raphson’s Method is faster than other methods. But needs more complicated conditions.(f(x) must be differentiable function, and f’(x) must be continuous.) Also, we need to find out f’(x) by ourselves.  
  
Fixed Point Method is slower than Newton-Raphson, But needs more simple conditions.(f(x) must be continuous.)  
  
Secant Method doesn’t need to calculate f’(x) by slightly changing Newtown-Raphson’s Method. Also, it tries to rise its accuracy by using error term(O(h))