STAT 428: Homework 6: Chapter 9 Optimization

Du, Yuting, yutingd3

Table of Contents

Please refer to the [**detailed homework policy document**] on Course Page for information about homework formatting, submission, and grading.

## Exercise 1

**Maximum Likelihood Estimator** In the exercise parts that follow, you must show derivation by hand(typed up in latex from within RMarkdown), include code and plots where appropriate. Also, do compare your results from code with those you got by hand.

Suppose that is a discrete random variable with

* ;
* ;
* ;
* ;

where is a parameter. The following 10 independent observations were taken from such a distribution: . In given 10 observations, there are 1 1’s, 3 2’s, 3 3’s and 3 4’s.

**(a)** What is the likelihood function ) for the sample ? $L(|X) = ( ) ( )^3 ( (1-))^3 ( (1-))^3 $

$L(|X) = ^4 (1-)^6 $

**(b)** What is the log-likelihood function ) for the sample ?

**(c)** What is the maximum log-likelihood estimate of ? *(Hint: Recall the optimize function in R.)*

$ I(|X) = 0$

f = function(x){  
 4\*log(x) + 6\*log(1-x)  
}  
  
result = optimize(f, c(0, 1), lower = 0, upper=1, maximum = TRUE)$maximum  
  
(result)

## [1] 0.4

## Exercise 2

**Maximum Likelihood Estimator in Logistic Regression** Suppose we have data in pairs for . Conditional on , is Bernoulli with success probability

The aim is to compute the maximum likelihood estimate $\hat{{\bf{\beta}}}$ of the parameter vector ${\bf{\beta}}=(\beta\_{0},\beta\_{1})^{T}$.

The log-likelihood is

$$
\ell({\bf{\beta}})=\sum\_{i=1}^{n}[y\_{i}log(p\_{i})+(1-y\_{i})log(1-p\_{i})]
$$

The data are given below:

X values:

1.34 -1.38 -0.19 -0.44 1.90 -0.80 0.91 0.26 1.37 -1.62 -0.96 1.90 0.99 1.96 -1.57 1.51 -1.61 -1.02 -0.92 -1.87 1.73 -1.23 -1.24 0.22 1.42 1.40 1.23 -0.75 1.47 -0.93

Y values:

1 0 0 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0

x = c(1.34, -1.38, -0.19, -0.44, 1.90, -0.80, 0.91, 0.26, 1.37, -1.62, -0.96, 1.90, 0.99, 1.96, -1.57, 1.51, -1.61, -1.02, -0.92, -1.87, 1.73, -1.23, -1.24, 0.22, 1.42)  
y = c(1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0)

**(a)** Use the function optim()to compute $\hat{{\bf{\beta}}}$ using initial value (.25,.75).

l <- function(beta) {  
 tmp <- exp(beta[1] + beta[2] \* x)  
 p <- tmp / (1 + tmp)  
 return(sum(y \* log(p) + (1 - y) \* log(1 - p)))  
}  
  
betahat <- optim(par=c(0.25, 0.75), fn=l, control = list(fnscale=-1))$par  
betahat

## [1] 0.1159 1.0286

**(b)** Again, starting with (.25,.75) find the next value when using the Newton-Raphson algorithm.

library(numDeriv)  
  
beta\_init <- c(0.25, 0.75)  
gradient <- grad(l, beta\_init)  
hess <- hessian(l, beta\_init)  
(nextbeta <- beta\_init - solve(hess) %\*% gradient)

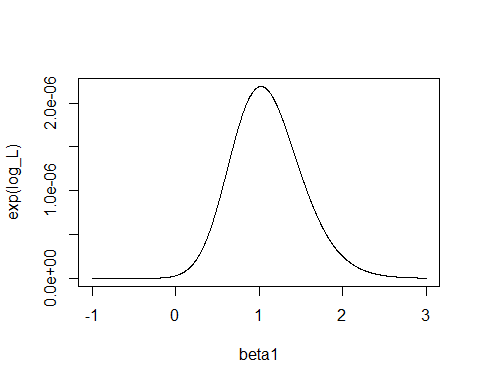
## [,1]  
## [1,] 0.1269  
## [2,] 0.9985

(nextbeta <- array(nextbeta))

## [1] 0.1269 0.9985

**(c)** Assume that , and plot the likelihood function as a function of .

l <- function(beta\_0, beta\_1) {  
 tmp <- exp(beta\_0 + beta\_1 \* x)  
 p <- tmp / (1 + tmp)  
 return(sum(y \* log(p) + (1 - y) \* log(1 - p)))  
}  
  
beta1 <- seq(-1, 3, length=1000)  
log\_L <- sapply(beta1, function(x) {l(0, x)})  
plot(beta1, exp(log\_L), type="l")



**(d)** Again, assume and compute using uniroot(), a grid search, and by the Newton-Raphson algorithm. You can use the plot in part (c) to find a good initial value.

# uniroot  
dl <- function(beta1) {  
 p <- exp(beta1 \* x) / (1 + exp(beta1 \* x))  
 grad <- sum(x \* (y - p))   
 return(grad)  
}  
(beta1\_uni <- uniroot(dl, c(0, 2))$root)

## [1] 1.022

# Grid Search  
grid <- seq(0, 2, 0.01)  
res <- sapply(grid, function(beta) {l(0, beta)})  
(beta1\_g <- grid[res == max(res)])

## [1] 1.02

# the Newton-Raphson algorithm  
Newton <- function(f, theres, x0, N){  
 h = 1e-4  
 i = 1  
 p = numeric(N)  
 while(i < N+1){  
 x1 <- (x0 - (f(x0) / ((f(x0 + h) - f(x0)) / h)))  
 p[i] <- x1  
 if (abs(x1 - x0) < theres) break  
 x0 <- x1  
 i <- i + 1  
 }  
 return(x0)  
}  
(beta1\_n <- Newton(dl, 1e-7, 0.8, 2000))

## [1] 1.022

## Exercise 3

**MLE with Survival Data**

The Weibull model can be used to model time until an event, such as the time until a battery burns out or a patient’s cancer returns.  
, where , the scale , the shape .

Consider the following (simulated) months until various batteries of the same type burn out:

Ex3 <- c(2.228, 2.051, 1.683, 3.285, 1.219, 2.879, 2.976, 2.112, 2.357, 2.425, 1.255, 2.562, 0.829, 2.581, 2.340, 3.043, 0.684, 1.810, 2.529, 0.729)

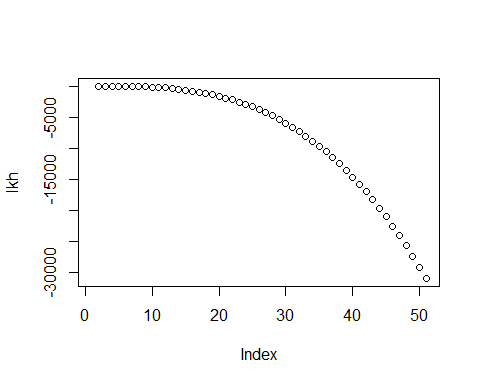
Assume we know that the time to burnout is Weibull, and that the shape .

1. We want to estimate the scale parameter . What is the log-likelihood for our sample?

Ln\_lkhd<-function(theta){  
 sum=0  
 for ( i in 1:length(Ex3)){  
 sum=sum+  
log(3)+3\*log(theta)+2\*log(Ex3[i])-  
(Ex3[i]\*theta)^3  
 }  
 return(sum)  
}

1. Plot the likelihood function as a function of .

lkh<-c()  
for ( theta in seq(0,5,.1)){  
 lkh<-c(lkh,Ln\_lkhd(theta))  
}  
plot(lkh)



1. What is the maximum log-likelihood estimate of ?

max\_est\_theta=  
(length(Ex3)/sum(Ex3^3))^(1/3)  
max\_est\_theta

## [1] 0.4314

## Exercise 4

Do exercise 9.3 in the book.

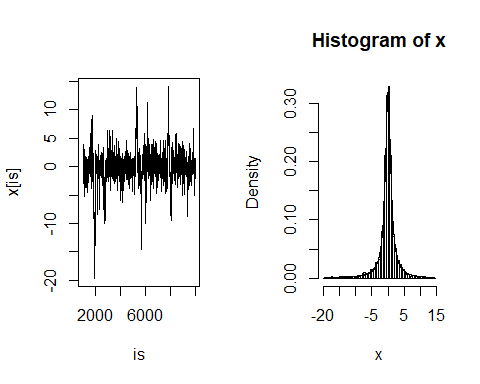
Exercise 9.3 Use the Metropolis-Hastings sampler to generate random variables from a standard Cauchy distribution. Discard the first 1000 of the chain, and compare the deciles of the generated observations with the deciles of the standard Cauchy distribution (see qcauchy or qt with df=1). Recall that a distribution has density function

The standard Cauchy has the density. (Note that the standard Cauchy density is equal to the Student density with one degree of freedom.)

theta = 1  
eta = 0  
N = 10000  
  
stopifnot(theta > 0)  
  
df = function(x) {  
 1/(theta\*pi\*(1+((x-eta)/theta)^2))  
}  
  
dg = function(x, df) {  
 dnorm(x = x, mean = df)  
}  
  
rg = function(df) {  
 rnorm(n = 1, mean = df)  
}  
  
mh = function (N, df, dg, rg) {  
 x = numeric(N)  
 x[1] = rg(1)  
 k = 0  
 u = runif(N)  
 for (i in 2:N) {  
 xt = x[i-1]  
 y = rg(xt)  
 r = df(y) \* dg(xt, y) / (df(xt) \* dg(y, xt))  
 if (u[i] <= r) {  
 x[i] = y  
 } else {  
 k = k + 1  
 x[i] = xt  
 }  
 }  
 print(k)  
 return(x)  
}  
  
x = mh(N, df, dg, rg)

## [1] 2329

is = 1001:N  
par(mfrow = c(1,2))  
plot(is, x[is], type="l")  
hist(x, probability = TRUE, breaks = 100)  
plot.x = seq(min(x), max(x), 0.01)  
lines(plot.x, df(plot.x))



par(mfrow = c(1,1))

## Exercise 5

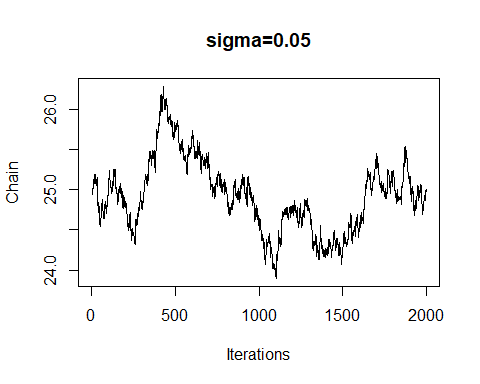
Do exercise 9.4 from the book.

Exercise 9.4 Implement a random walk Metropolis sampler for generating the standard Laplace distribution (See Exercise 3.2). For the increment, simulate from a normal distribution. Compare the chains generated when different variances are used for the proposed distribution. Also, compute the acceptance rates of each chain.

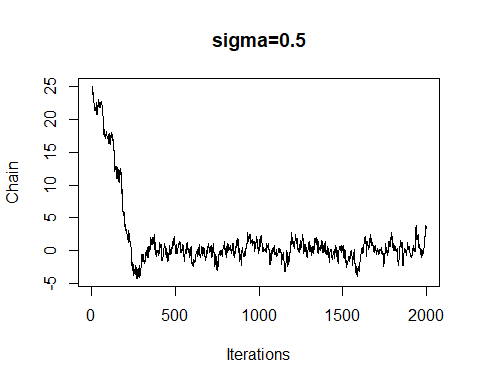
Lden=function(x){  
 return (1/2\*exp(-abs(x)))  
}  
  
rw.Metropolis <- function(sigma, x0, N) {  
 x <- numeric(N)  
 x[1] <- x0  
 u <- runif(N)  
 k <- 0  
 for (i in 2:N) {  
 y <- rnorm(1, x[i-1], sigma)  
 if (u[i] <= (Lden(y) / Lden((x[i-1])))){  
 x[i] <- y  
 k=k+1  
 }  
 else {  
 x[i] <- x[i-1]  
 }  
 }  
 return(list(x=x, acrate=k/N))  
}  
  
N <- 2000  
sigma <- c(0.05, 0.5, 2, 16)  
x0 <- 25 # starting  
rw1 <- rw.Metropolis(sigma[1], x0, N)  
rw2 <- rw.Metropolis(sigma[2], x0, N)  
rw3 <- rw.Metropolis(sigma[3], x0, N)  
rw4 <- rw.Metropolis(sigma[4], x0, N)  
# number of rejected candidate points  
print(c(rw1$acrate, rw2$acrate, rw3$acrate, rw4$acrate))

## [1] 0.9730 0.8240 0.5210 0.1035

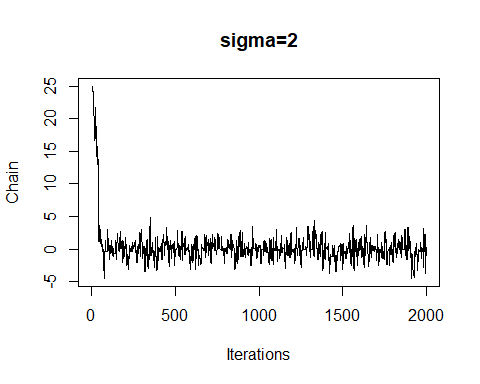
# plot of chain  
plot(rw1$x,type='l',ylab='Chain',xlab='Iterations',main='sigma=0.05')



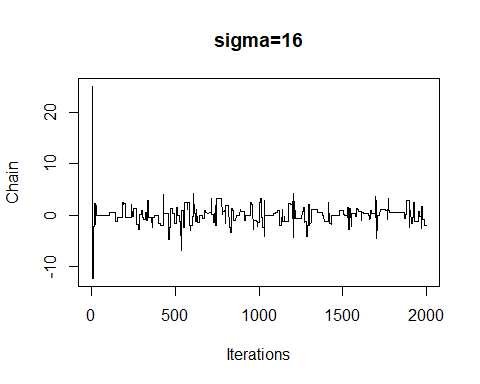
plot(rw2$x,type='l',ylab='Chain',xlab='Iterations',main='sigma=0.5')



plot(rw3$x,type='l',ylab='Chain',xlab='Iterations',main='sigma=2')



plot(rw4$x,type='l',ylab='Chain',xlab='Iterations',main='sigma=16')



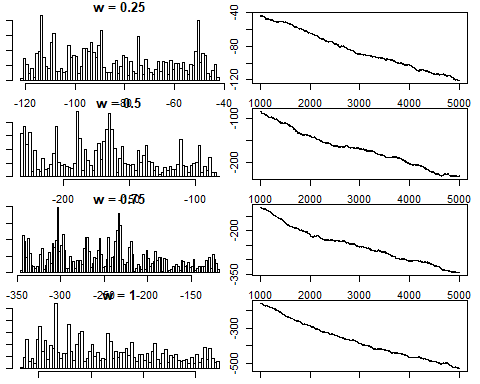
‘sigma=2’ seems best here.

## Exercise 6

Do exercise 9.5 from the book.

Exercise 9.5 What effect, if any, does the width have on the mixing of the chain in Example 9.5? Repeat the simulation keeping the random number seed fixed, trying different proposal distribution based on the random increments from , varying .

b = 0.2  
b.lim = c(0, 0.5)  
days = 250  
secs = 1:5  
m = 5000  
burn = 1000  
ws = c((1:4)/4)  
  
prob.vector = function (b) {  
 return(c(1/3, (1-b)/3, (1-2\*b)/3, 2\*b/3, b/3))  
}  
ps = prob.vector(b)  
i = sample(secs, size = days, prob = ps, replace = TRUE)  
win = tabulate(i)  
  
  
  
posterior = function (x, win) {  
 if (x < b.lim[1] || x > b.lim[2]) {  
 return(0)  
 }  
 nums = sapply(split(1:days, rep(1:(length(win)), days/length(win))), function(n) prod(n))  
 dens = sapply(win, function(w) factorial(w))  
 probs = prob.vector(x) ^ win  
 return(prod(nums/dens\*probs))  
}  
  
  
prob.ratio = function (n, d, win) {  
 return(prod(prob.vector(n)^win / prob.vector(d)^win))  
}  
  
rw.b = function (w) {  
 x = numeric(m)  
   
 u = runif(m)  
 v = runif(m, -w, w)  
 x[1] = w  
 for (i in 2:m) {  
 xt = x[i-1]  
 y = xt + v[i]  
 r = prob.ratio(y, xt, win)  
 if (u[i] <= r) {  
 x[i] = y  
 } else {  
 x[i] = xt  
 }  
 }  
   
 return(x)  
}  
  
xbs = lapply(ws, function(w) rw.b(w))  
par(mfrow = c(length(ws), 2))  
par(mar=c(1,1,1,1))#debug for figure image too large  
is = (burn+1):m  
for (i in 1:length(ws)) {  
 xb = xbs[[i]]  
 xb = xb[is]  
 xb.seq = seq(min(xb), max(xb), 0.05)  
 hist(xb, breaks = 100, probability = TRUE, main = paste('w = ', ws[i], sep = ''))  
  
 lines(xb.seq, sapply(xb.seq, function(x) posterior(x, win)))  
 plot(is, xb, type="l")  
}



par(mfrow = c(1,1))

## Exercise 7

Do exercise 9.6 from the book.

Exercise 9.6 Rao [220, Sec. 5g] presented an example on genetic linkage of 197 animals in four categories (also discussed in [67, 106, 171, 266]). The group sizes are (125, 18, 20, 34). Assume that the probabilities of the corresponding multinomial distribution are

Estimate the posterior distribution of given the observed sample, using one of the methods in this chapter.

sizes = c(125, 18, 20, 34)  
size = sum(sizes)  
  
m = 10000  
burn = 2000  
is = (burn+1):m  
  
prob.vector = function (theta) {  
 return(c(2 + theta, (1-theta), (1-theta), theta) / 4)  
}  
  
prob.ratio = function (n, d) {  
  
 return(prod(prob.vector(n)^sizes / prob.vector(d)^sizes))  
}  
  
  
  
x.rw = numeric(m)  
k.rw = 0  
u = runif(m)  
v = runif(m, -0.25, 0.25)  
x.rw[1] = v[1]  
for (i in 2:m) {  
 xt = x.rw[i-1]  
 y = xt + v[i]  
 r = min(prob.ratio(y, xt), 1)  
  
 if (!is.nan(r) && u[i] <= r) {  
 x.rw[i] = y  
 } else {  
 k.rw = k.rw + 1  
 x.rw[i] = xt  
 }  
}  
  
  
  
sd = 0.5  
min = -0.8  
max = 0.8  
  
rg = function(p) {  
 return(runif(1, min - abs(p), max + abs(p)))  
  
}  
  
dg = function(x, p) {  
 return(dunif(x, min - abs(p), max + abs(p)))  
  
}  
  
x.mh = numeric(m)  
k.mh = 0  
u = runif(m)  
x.mh[1] = rg(0)  
for(i in 2:m) {  
 xt = x.mh[i-1]  
 y = rg(xt)  
 r = min(prob.ratio(y, xt) \* dg(xt, y) / dg(y, xt), 1)  
 if (!is.na(r) && u[i] <= r) {  
 x.mh[i] = y  
 } else {  
 x.mh[i] = xt  
 k.mh = k.mh + 1  
 }  
}  
  
  
  
x.i = numeric(m)  
x.i[1] = rg(0)  
k.i = 0  
u = runif(m)  
for (i in 2:m){  
 xt = x.i[i-1]  
 y = rg(0)  
 r = prob.ratio(y, xt) \* dg(xt, 0)/dg(y, 0)  
 if (u[i] <= r) {  
 x.i[i] = y  
 } else {  
 x.i[i] = xt  
 k.i = k.i + 1  
 }  
}  
  
print(k.rw)

## [1] 6732

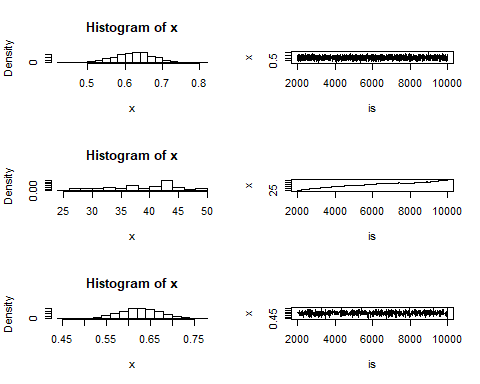
print(k.mh)

## [1] 9828

print(k.i)

## [1] 8918

par(mfrow = c(3,2))  
  
xs = as.list(x.rw, x.mh)  
  
x = x.rw[is]  
hist(x, probability = TRUE)  
plot(is, x, type='l')  
  
x = x.mh[is]  
hist(x, probability = TRUE)  
plot(is, x, type='l')  
  
x = x.i[is]  
hist(x, probability = TRUE)  
plot(is, x, type='l')



par(mfrow = c(1,1))

## Exercise 8

Do exercise 9.10 from the book.

Exercise 9.10 Refer to Example 9.1. Use the Gelman-Rubin method to monitor convergence of the chain, and run the chain until the chain has converged approximately to the target distribution according to . (See Exercise 9.9.) Also use the coda [212] package to check for convergence of the chain by the Gelman-Rubin method. Hints: See the help topics for the coda functions gelman.diag, gelman.plot, as.mcmc, and mcmc.list.

#Defining the function to calculate the G-R statistic  
Gelman.Rubin <- function(psi){  
  
 psi <- as.matrix(psi)  
 n <- ncol(psi)  
 k <- nrow(psi)  
 psi.means <- rowMeans(psi)   
 B <- n \* var(psi.means)   
 psi.w <- apply(psi, 1, "var")   
 W <- mean(psi.w)   
 v.hat <- W\*(n-1)/n + (B/n)   
 r.hat <- v.hat / W   
 return(r.hat)  
}  
  
f <- function(x, sigma){   
 if (any(x < 0)) return (0)  
 stopifnot(sigma > 0)  
 return((x / sigma^2) \* exp(-x^2 / (2\*sigma^2)))  
}  
  
k <- 4   
sigma <- 4  
x <- as.matrix(c(0.01, 2, 4, 6))  
ral.chain <- function(x){  
 xi <- numeric(nrow(x))  
 for(i in 1:length(xi)){  
 xt <- x[i, ncol(x)]  
 y <- rchisq(1, df = xt)   
 # Using chi square distribution  
 num <- f(y, sigma) \* dchisq(xt, df = y)  
 den <- f(xt, sigma) \* dchisq(y, df = xt)  
 u <- runif(1)  
 if (u <= num/den) xi[i] <- y else{  
 xi[i] <- xt #y is rejected  
 }  
 }  
 return(cbind(x,xi))  
}  
r.hat = 10  
while(r.hat >= 1.2){  
 x <- ral.chain(x)  
 psi <- t(apply(x, 1, cumsum))  
 for (i in 1:nrow(psi))  
 psi[i,] <- psi[i,] / (1:ncol(psi))  
 r.hat <- Gelman.Rubin(psi)  
}  
b <- ncol(x)  
  
# check convergence by the G-R method  
library(coda)  
x.mcmc <- mcmc.list(as.mcmc(x[1,]),as.mcmc(x[2,]),as.mcmc(x[3,]),as.mcmc(x[4,]))  
geweke.diag(x.mcmc)

## [[1]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## var1   
## -2.637   
##   
##   
## [[2]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## var1   
## -0.4212   
##   
##   
## [[3]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## var1   
## -0.2442   
##   
##   
## [[4]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## var1   
## 0.7841

geweke.plot(x.mcmc)

