Assignment2

Xueying Zhou

1/29/2018

Problem1

(a)

$$L(\theta) = p_1 p_2 \ldots p_n \tag{1}$$
$$\tag{2}$$
$$lnL(\theta) = ln(p_1) + \cdots + p_n \tag{3}$$
$$\tag{4}$$
$$l(\theta) = -\sum_{i=1}^{n} ln\pi - \sum_{i=1}^{n} ln[1 + (x_i - \theta)^2] \tag{5}$$
$$\tag{6}$$
$$l(\theta) = -nln\pi - \sum_{i=1}^{n} ln[1 + (x_i - \theta)^2] \tag{7}$$
$$\tag{8}$$
$$l'(\theta) = -0 - \sum_{i=1}^{n} \frac{2(\theta - x_i)}{1 + (\theta - x_i)^2} \tag{9}$$
$$\tag{10}$$
$$l'(\theta) = -2\sum_{i=1}^{n} \frac{(\theta - x_i)}{1 + (\theta - x_i} \tag{11}$$
$$\tag{12}$$
$$l''(\theta) = -2\sum_{i=1}^{n} \frac{1 + (\theta - x_i)^2 - 2(\theta - x_i)^2}{[1 + (\theta - x_i)^2]^2} \tag{13}$$
$$\tag{14}$$
$$l''(\theta) = -2\sum_{i=1}^{n} \frac{1 - (\theta - x_i)^2}{[1 + (\theta - x_i)^2]^2} \tag{15}$$
$$\tag{16}$$
$$p'(x) = -\frac{2(x - \theta)}{\pi[1 + (x - \theta)^2]^2} \tag{17}$$
$$\tag{18}$$
$$I(\theta) = \frac{4n}{\pi} \int_{-\infty}^{\infty} \frac{(x - \theta)^2}{[1 + (x - \theta)^2]^3} dx \tag{19}$$
$$\tag{20}$$
$$I(\theta) = \frac{4n}{\pi} \int_{-\infty}^{\infty} \frac{x^2}{[1 + x^2]^3} dx \tag{21}$$
$$\tag{22}$$
$$\tag{23}$$

if we choose $x = tanx$, $1 + x^2 = \frac{1}{cos^2\theta}$,

$$I(\theta) = \frac{4n}{\pi}\int_{-\infty}^{\infty}\frac{sin^2x}{cos^2x}cos^6xd(tanx) \tag{24}$$

$$\tag{25}$$

$$I(\theta) = \frac{4n}{\pi}\int_{-\infty}^{\infty}\frac{1}{cos^2x}sin^2xcos^4xdx \tag{26}$$

$$\tag{27}$$

$$I(\theta) = \frac{4n}{\pi}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}}\frac{sin^2 2x}{4}dx \tag{28}$$

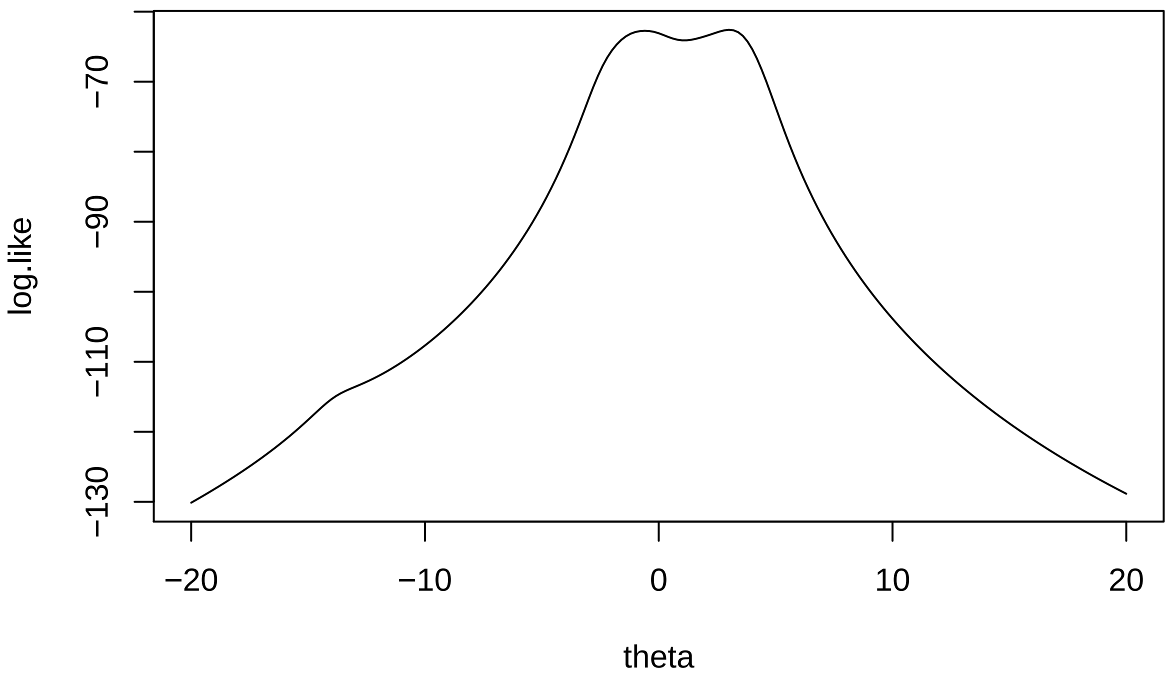
$$\tag{29}$$

$$I(\theta) = \frac{4n}{\pi}\frac{\pi}{8} = n/2 \tag{30}$$

(b)

First, we graph the log-likelihood funtion $l(\theta) = -nln\pi - \sum_{i=1}^{n} ln[1+(x_i-\theta)^2]$ as shown in the former section

```
x <- c(1.77, -0.23, 2.76, 3.80, 3.47, 56.75, -1.34, 4.24, -2.44,
       3.29, 3.71, -2.40, 4.53, -0.07, -1.05, -13.87, -2.53, -1.75)
#theta <- c(-11, -1, 0, 1.5, 4, 4.7, 7, 8, 38)
theta <- seq(from = -20, to = 20, by = 0.2)
i <- 1
log.like <- rep(0, 201)
for (i in 1:201) {
  log.like[i] <- -18*log(pi) - sum(log(1 + (theta[i] - x)^2))
}
plot(theta, log.like, type = "l")
```



Then we find the MLE for θ through application of the Newton-Raphson method for the nine given starting points. By setting the the sample mean, which turns out to be 3.257778, as the starting point

```r
library(elliptic)
```

```
##
## Attaching package: 'elliptic'
```

```
## The following objects are masked from 'package:stats':
##
##     sd, sigma
```

```
## The following object is masked from 'package:base':
##
##     is.primitive
```

```r
x <- c(1.77, -0.23, 2.76, 3.80, 3.47, 56.75, -1.34, 4.24, -2.44,
       3.29, 3.71, -2.40, 4.53, -0.07, -1.05, -13.87, -2.53, -1.75)
f <- function(theta){ - 2 * sum((theta - x) / (1 + (theta - x)^2))}
fdash <- function(theta){ - 2 * sum((1 - (theta - x)^2) / ((1 + (theta - x)^2))^2)}

s.p <- c(-11, -1, 0, 1.5, 4, 4.7, 7, 8, 38)

for (i in 1:9) {
    result <- newton_raphson(s.p[i], f, fdash, maxiter=100)
  print(paste0("starting point is: ", s.p[i]))
  print(result)
}
```

```
## [1] "starting point is: -11"
## $root
## [1] -2.080688e+17
##
## $f.root
## [1] 1.730197e-16
##
## $iter
## [1] 55
##
## [1] "starting point is: -1"
## $root
## [1] -0.5914735
##
## $f.root
## [1] 3.469447e-17
##
## $iter
## [1] 4
##
## [1] "starting point is: 0"
## $root
## [1] -0.5914735
##
## $f.root
## [1] -1.873501e-16
##
## $iter
## [1] 5
```

```
##
## [1] "starting point is: 1.5"
## $root
## [1] 1.09273
##
## $f.root
## [1] -1.387779e-17
##
## $iter
## [1] 5
##
## [1] "starting point is: 4"
## $root
## [1] 3.021345
##
## $f.root
## [1] -6.938894e-17
##
## $iter
## [1] 4
##
## [1] "starting point is: 4.7"
## $root
## [1] -0.5914735
##
## $f.root
## [1] -1.873501e-16
##
## $iter
## [1] 5
##
## [1] "starting point is: 7"
## $root
## [1] 1.736482e+17
##
## $f.root
## [1] -2.073157e-16
##
## $iter
## [1] 56
##
## [1] "starting point is: 8"
## $root
## [1] 1.838207e+17
##
## $f.root
## [1] -1.95843e-16
##
## $iter
## [1] 57
##
## [1] "starting point is: 38"
## $root
## [1] 3.226576e+17
```

```
##
## $f.root
## [1] -1.115734e-16
##
## $iter
## [1] 54
```

```r
mean(x)
```

```
## [1] 3.257778
```

```r
#sample mean 3.257778
newton_raphson(3.257778, f, fdash, maxiter=100)
```

```
## $root
## [1] 3.021345
##
## $f.root
## [1] -6.938894e-17
##
## $iter
## [1] 5
```

As shown in the results, taking the sample mean 3.257778 as the starting point gives the root 3.021345 while the other trials have more obvious difference between the starting point and the root. Therefore, the sample mean is a good starting point. Better choice might be available though.

(c)

We define the function for the fixed-point iteration with an accuracy tolerance of 10^{-6} and the maximum iteration times to be 1000.

```r
p.fixed <- function(p0,alpha,obs,tol = 1E-6,max.iter = 1000,verbose=F){
  pold <- p0
   pnew <- pold + alpha * (-2) * sum((pold - obs)/(1 + (pold - obs)^2))
   iter <- 1
  while ((abs(pnew - pold) > tol) && (iter < max.iter)){
    pold <- pnew
    pnew <-  pold + alpha * (-2) * sum((pold - obs)/(1 + (pold - obs)^2))
    iter <- iter + 1
    if(verbose)
      cat("At iteration", iter, "value of p is:", pnew, "\n")
  }
  if (abs(pnew - pold) > tol) {
    cat("Algorithm failed to converge")
    return(NULL)
  }
  else {
    cat("Algorithm converged, in :" ,iter,"iterations \n")
    return(pnew)
  }
}

x <- c(1.77, -0.23, 2.76, 3.80, 3.47, 56.75, -1.34, 4.24, -2.44,
       3.29, 3.71, -2.40, 4.53, -0.07, -1.05, -13.87, -2.53, -1.75)
```

```r
p.fixed(p0 = -1, alpha = 1.00, obs = x)
```

```
## Algorithm failed to converge
```

```
## NULL
```

```r
p.fixed(p0 = -1, alpha = 0.64, obs = x)
```

```
## Algorithm converged, in : 11 iterations
```

```
## [1] -0.5914734
```

```r
p.fixed(p0 = -1, alpha = 0.25, obs = x)
```

```
## Algorithm converged, in : 20 iterations
```

```
## [1] -0.5914741
```

```r
#alpha = 1 will not converge.

s.p <- c(-11, -1, 0, 1.5, 4, 4.7, 7, 8, 38)
alpha <- c(1, 0.64, 0.25)
i <- 1
j <- 1
for (i in 1:9){
  for (j in 1:3){
   print(paste0("p0 = ", s.p[i], " scaling alpha =", alpha[j]))
    print(p.fixed(p0 = s.p[i], alpha = alpha[j], obs = x))
  }
}
```

```
## [1] "p0 = -11 scaling alpha =1"
## Algorithm converged, in : 218 iterations
## [1] -0.591474
## [1] "p0 = -11 scaling alpha =0.64"
## Algorithm converged, in : 17 iterations
## [1] -0.5914734
## [1] "p0 = -11 scaling alpha =0.25"
## Algorithm converged, in : 28 iterations
## [1] -0.5914745
## [1] "p0 = -1 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = -1 scaling alpha =0.64"
## Algorithm converged, in : 11 iterations
## [1] -0.5914734
## [1] "p0 = -1 scaling alpha =0.25"
## Algorithm converged, in : 20 iterations
## [1] -0.5914741
## [1] "p0 = 0 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = 0 scaling alpha =0.64"
## Algorithm converged, in : 11 iterations
## [1] -0.5914736
## [1] "p0 = 0 scaling alpha =0.25"
## Algorithm converged, in : 20 iterations
## [1] -0.5914726
## [1] "p0 = 1.5 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = 1.5 scaling alpha =0.64"
## Algorithm failed to convergeNULL
## [1] "p0 = 1.5 scaling alpha =0.25"
## Algorithm converged, in : 12 iterations
## [1] 3.021345
## [1] "p0 = 4 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = 4 scaling alpha =0.64"
## Algorithm converged, in : 12 iterations
## [1] -0.5914736
## [1] "p0 = 4 scaling alpha =0.25"
## Algorithm converged, in : 8 iterations
## [1] 3.021345
## [1] "p0 = 4.7 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = 4.7 scaling alpha =0.64"
## Algorithm converged, in : 13 iterations
## [1] -0.5914736
## [1] "p0 = 4.7 scaling alpha =0.25"
## Algorithm converged, in : 9 iterations
## [1] 3.021345
## [1] "p0 = 7 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = 7 scaling alpha =0.64"
## Algorithm failed to convergeNULL
## [1] "p0 = 7 scaling alpha =0.25"
## Algorithm converged, in : 8 iterations
## [1] 3.021345
## [1] "p0 = 8 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = 8 scaling alpha =0.64"
## Algorithm converged, in : 14 iterations
## [1] -0.5914736
## [1] "p0 = 8 scaling alpha =0.25"
## Algorithm converged, in : 10 iterations
## [1] 3.021345
## [1] "p0 = 38 scaling alpha =1"
## Algorithm failed to convergeNULL
## [1] "p0 = 38 scaling alpha =0.64"
## Algorithm failed to convergeNULL
## [1] "p0 = 38 scaling alpha =0.25"
## Algorithm converged, in : 93 iterations
## [1] 3.021345
```

(d)

Note that to maximize the log-likelihood is equivalent to minimise its negative value. We first use the function *nlminb* with the fisher score

$$I(\theta) = n/2$$

as found in part (a) to find the MLE for θ.

```r
x <- c(1.77, -0.23, 2.76, 3.80, 3.47, 56.75, -1.34, 4.24, -2.44,
       3.29, 3.71, -2.40, 4.53, -0.07, -1.05, -13.87, -2.53, -1.75)
s.p <- c(-11, -1, 0, 1.5, 4, 4.7, 7, 8, 38)

i <- 1

for (i in 1:9) {
  f <- function(theta){length(x) * log(pi) + sum(1 + (theta - x)^2)}
  grf <- function(theta){2 * sum((theta - x) / (1 + (theta - x)^2))}
  fs <- function(theta){matrix(length(x) / 2, nrow = 1)}
  z <- nlminb(s.p[i], f, grf,fs)}
```

Then we refine the estimation by running Newton-Raphson method as we did in part (b).

```r
  f <- function(theta){-2*sum((theta - x)/(1 + (theta - x)^2))}
  fdash <- function(theta){-2*sum((1 - (theta - x)^2)/((1 + (theta - x)^2))^2)}
  result <- newton_raphson(z$par, f, fdash, maxiter=1000)
  print(paste0("starting point =", s.p[i]))
```

```
## [1] "starting point =38"
```

```r
  print(paste0("Fisher scoring"))
```

```
## [1] "Fisher scoring"
```

```r
  print(z)
```

```
## $par
## [1] 19.17511
##
## $objective
## [1] 7945.465
##
## $convergence
## [1] 1
##
## $iterations
## [1] 150
##
## $evaluations
## function gradient
##      151      151
##
## $message
## [1] "iteration limit reached without convergence (10)"
```

```r
  print(paste0("Newton-Rasphon"))
```

```
## [1] "Newton-Rasphon"
```

```r
  print(result)
```

```
## $root
## [1] 2.889657e+17
##
## $f.root
## [1] -1.245823e-16
```

```
##
## $iter
## [1] 55
```

(e) From the results, we can see from the numbers of iteration that the fixed-point method is less efficient thanFisher scoring with Newton-Rasphon in this case. As for the stability,the later one performs better as well.

Problem2

(a)

First, we find the log-likelihood function of θ

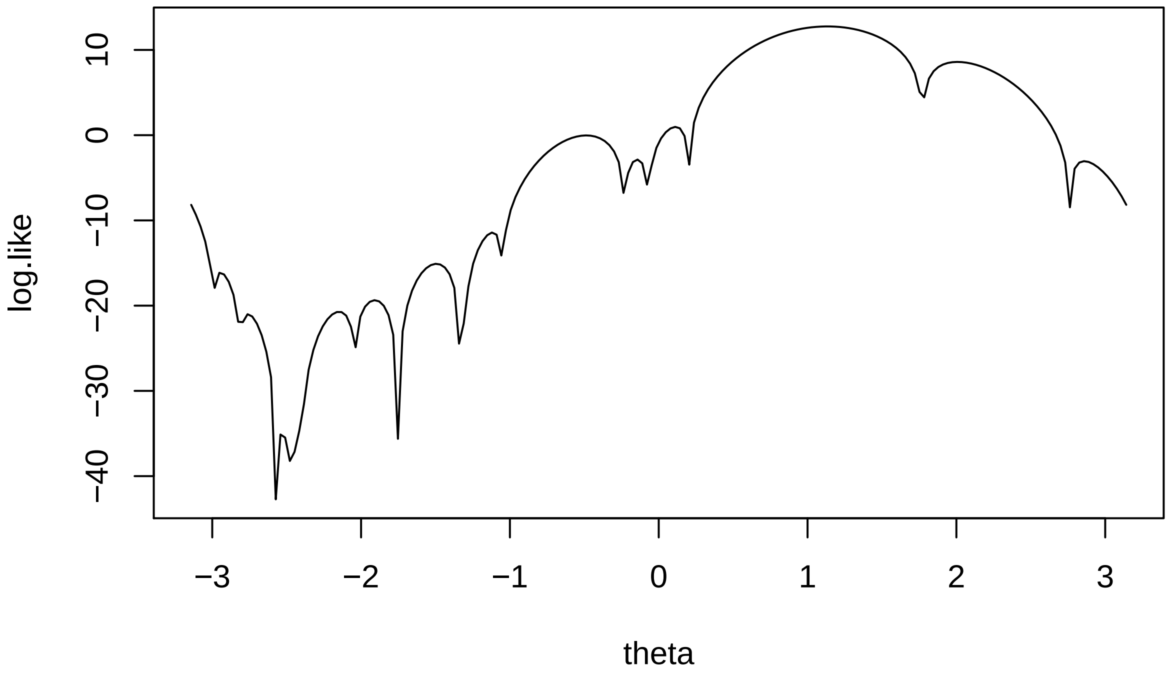
$$L(\theta) = p_1 p_2 ... p_{19}$$

$$lnL(\theta) = lnp_1 + \cdots + p_{19}$$

$$l(\theta) = \sum_{i=1}^{19} ln(\frac{1 - cos(x-\theta)}{2\pi})$$

Then, we plot the graph for the $l(\theta)$ as required

```r
theta <- seq(from = -pi, to = pi, by = 2 * pi / 199)
i <- 1
log.like <- rep(0, 200)
for (i in 1:200) {
  log.like[i] <- sum(log((1 - cos(x - theta[i])) / 2 * pi))
}
plot(theta,log.like,type = "l")
```



(b)

We solve for the value of the estimator $\hat{\theta}_{moment}$

$$E[x|\theta] = \int_0^{2\pi} x\frac{1-cos(x-\theta)}{2\pi} \tag{31}$$

$$\tag{32}$$

$$E[x|\theta] = \frac{1}{2\pi}[\int_0^{2\pi} xdx - \int_0^{2\pi} xcos(x-\theta)dx)] \tag{33}$$

$$\tag{34}$$

$$E[x|\theta] = \pi - \frac{1}{2\pi}(xsin(x-\theta) + cos(x-\theta))|_0^{2\pi} \tag{35}$$

$$\tag{36}$$

$$E[x|\hat{\theta}_{moment}] = \pi + sin(\hat{\theta}_{moment}) = \bar{x} \tag{37}$$

$$\tag{38}$$

$$\hat{\theta}_{moment} = arcsin(\bar{x} - \pi) \tag{39}$$

$$\tag{40}$$

$$\tag{41}$$

Therefore, the obeject function is $f(\theta) = \pi + sin(\hat{\theta}_{moment}) - \bar{x}$. Here we use the function *uniroot*

```
mean(x)
```

```
## [1] 3.257778
```

```
f1<-function(theta){
  pi+sin(theta)-mean(x)
}

uniroot.1<-uniroot(f1,lower=-pi,upper=-pi/2,extendInt = "yes")$root[1]
uniroot.2<-uniroot(f1,lower=-pi/2,upper=0,extendInt = "yes")$root[1]
uniroot.3<-uniroot(f1,lower=0,upper=pi/2,extendInt = "yes")$root[1]
uniroot.4<-uniroot(f1,lower=pi/2,upper=pi,extendInt = "yes")$root[1]

theta.mom <- c(uniroot.1, uniroot.2, uniroot.3, uniroot.4)
return(theta.mom)
```

```
## [1] -3.2580418 -3.2580413  0.1164476  3.0251450
```

Roots are found as -3.2580418, -3.2580413, 0.1164476 and 3.0251450. ##(c) To use the Newton-Raphson method, similar to the earlier process, we will need the object function of θ, which is

$$f(\theta) = \sum_{i=1}^{19} \frac{sin(\theta - x_i)}{1 - cos(\theta - x_i)}$$

, and its first derivative

$$f'(\theta) = \sum_{i=1}^{19} \frac{1}{cos(\theta - x_i) - 1}$$

```
library(elliptic)
```

```r
x <- c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96,
       2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, 0.52)

f<-function(theta){
  sum( sin(theta-x)/(1-cos(theta-x)) )
}
fdash<-function(theta){
  sum( 1 / (cos(theta-x)-1))
}

new.raphson <- function (initial, f, fdash, maxiter, give = TRUE, tol = .Machine$double.eps)
{
  old.guess <- initial
  for (i in seq_len(maxiter)) {
    new.guess <- old.guess - f(old.guess)/fdash(old.guess)
    jj <- f(new.guess)
    if (is.na(jj) | is.infinite(jj)) {
      break
    }
    if (near.match(new.guess, old.guess) | abs(jj) < tol) {
      if (give) {
        return(list(root = new.guess, f.root = jj, iter = i))
      }
      else {
        return(new.guess)
      }
    }
    old.guess <- new.guess
  }
  return("did not converge")
}

for (i in 1:length(theta.mom) ){
print(new.raphson(theta.mom[i],f,fdash,maxiter=1000))
}
```

```
## $root
## [1] -3.112471
##
## $f.root
## [1] 1.765255e-14
##
## $iter
## [1] 8
##
## $root
## [1] -3.112471
##
## $f.root
## [1] 1.765255e-14
##
## $iter
## [1] 8
##
```

```
## [1] "did not converge"
## $root
## [1] 3.170715
##
## $f.root
## [1] -3.719247e-15
##
## $iter
## [1] 8
```

Applying the values of $\hat{\theta}_{moment}$ found in part (b), the results -3.112471, -3.114472, 3.170715 were obtained for each value. Note that if start at 0.1164476, it won't converge. ##(d)

```
newton_raphson(-2.7,f,fdash,maxiter=100)
```

```
## $root
## [1] -2.668857
##
## $f.root
## [1] -2.926999e-13
##
## $iter
## [1] 5
```

```
newton_raphson(2.7,f,fdash,maxiter=100)
```

```
## $root
## [1] 2.848415
##
## $f.root
## [1] -3.008704e-14
##
## $iter
## [1] 6
```

(e)

```
N_R <- function (initial, f, fdash, maxiter, give = TRUE, tol = .Machine$double.eps)
{
  old.guess <- initial
  for (i in seq_len(maxiter)) {
    new.guess <- old.guess - f(old.guess)/fdash(old.guess)
    jj <- f(new.guess)
    if (is.na(jj) | is.infinite(jj)) {
      break
    }
    if (near.match(new.guess, old.guess) | abs(jj) < tol) {
      if (give) {
        return(list(root = new.guess, f.root = jj, iter = i))
      }
      else {
        return(new.guess)
      }
    }
    old.guess <- new.guess
  }
  return(list(root = "Failed to Converge", f.root = jj, iter = i))
}

s.p <- seq(from = -pi, to = pi, by = 2*pi/199)

newton_raphson(s.p[1],f,fdash,maxiter=1000)
```

```
## $root
## [1] -3.112471
##
## $f.root
## [1] 1.765255e-14
##
## $iter
## [1] 5
```

```
newton_raphson(s.p[2],f,fdash,maxiter=1000)
```

```
## $root
## [1] -3.112471
##
## $f.root
## [1] 1.765255e-14
##
## $iter
## [1] 4
```

```
newton_raphson(s.p[3],f,fdash,maxiter=1000)
```

```
## $root
## [1] -3.112471
##
## $f.root
## [1] 1.765255e-14
##
## $iter
## [1] 5
```

```
out <- data.frame(
  start.point <- s.p[1:200],
  root <- rep(0,200)
)
names(out) <- c("start.point","root")


for(i in 1:200) {
  result <- N_R(s.p[i],f,fdash,maxiter=1000)
  out[i,2] <- result$root
}

out$root <-as.factor(out$root)
#There are 18 levels
i <- 1
for (i in 1:18){
  subgrp <- data.frame(
    start.point <- rep(0,length(which(out$root == levels(out$root)[i]))),
    root <- rep(0,length(which(out$root == levels(out$root)[i])))
  )
  names(subgrp) <- c("start.point","root")
  subgrp$start.point <- out[which(out$root == levels(out$root)[i]),1]
  subgrp$root <- out[which(out$root == levels(out$root)[i]),2]
  assign(paste0("root.",i), subgrp)
}
```

Problem3

(a)

Since it is given that

$$N_t = \frac{KN_0}{N_0 + (K - N_0)exp(-rt)}$$

with $N_0 = 2$, we have

$$N_t = \frac{2K}{2 + (K - 2)exp(-rt)}$$
$$exp(-rt) = \frac{2(K - N_t)}{N_t(K - 2)}$$
$$rt = ln\frac{N_t(K-2)}{2(K - N_t)}$$
$$r = ln\frac{N_t(K-2)}{2(K - N_t)}\frac{1}{t}$$

Now we rename the unknown parameter K that represents the population carrying capacity of the environment as θ_1, and the unknown growth rate r as θ_2. It has been shown in (a) that the error

$$f(N_t, r) = N_t - \frac{2K}{2 + (K - 2)exp(-rt)}$$

The Nonlinear Lesat Squares function is called to minimize the squared error as required. The results show that with the initial value $(K, r) = (1200, 0.1716788)$, the optimization will be reached with $(K, r) = (1049.4069185, 0.1182685)$

```
beetles <- data.frame(
  days = c(0, 8, 28, 41, 63, 69, 97, 117, 135, 154),
  beetles = c(2, 47, 192, 256, 768, 896, 1120, 896, 1184, 1024))

K <- 1200
r.t <- log((beetles$beetles*(K-2))/(K - beetles$beetles)*2)
r.series <- r.t / beetles$days
mean(r.series[2:10])
```

```
## [1] 0.1716788
```

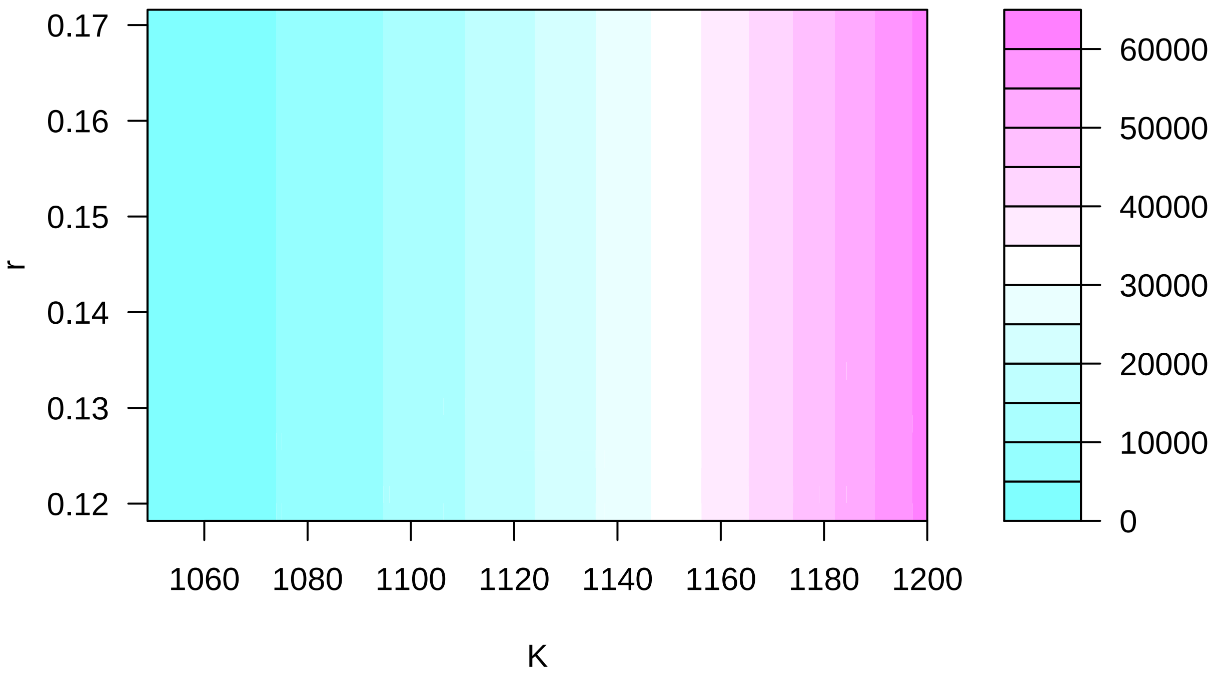
```
#r <- 0.1716788

#theta1 <- N
#theta2 <- r
#beetles ~ theta1*2/(2+(theta1-2)exp(-1*theta2*days))
pop.mod <- nls(beetles ~ N*2/(2+(N-2)*exp((-r)*days)),start=list(N = 1200, r = 0.1716788),data=beetles,
## 677146.3 :  1200.0000000    0.1716788
## 110309.7 :  988.5766717   0.1367234
## 76986.61 :  1024.1437803    0.1224583
## 73503.58 :  1045.8468846    0.1189023
## 73421.83 :  1048.9806817    0.1183764
## 73419.76 :  1049.3390375    0.1182867
## 73419.7 :  1049.3958171    0.1182715
## 73419.7 :  1049.4053152    0.1182689
## 73419.7 :  1049.4069185    0.1182685
```

```
#Gaussian-Newton
i <- 1
j <- 1
t.day<- as.numeric(beetles$days)
n.b <- as.numeric(beetles$beetles)
theta <- matrix(c(1200, 0.1716788), nrow = 2)

for (j in 1:10) {
  error.matrix0 <- c()
  for (i in 1:10){
    t <- t.day[i]
    n <- n.b[i]
    K <- theta[1,1]
    r <- theta[2,1]
    error.matrix <- matrix( n - (K*2)/(2+(K-2)*exp(-r*t)), nrow = 1)
    error.matrix0 <- rbind(error.matrix0, t(error.matrix))
  }

  z.matrix <- error.matrix0
  sse <- sum(z.matrix[1:10,1]^2)

  i <- 1
  theta.note <- c("K","r")
  gradient.matrix0 <- c()
  for (i in 1:10){
    t <- t.day[i]
    n <- n.b[i]
    K <- theta[1,1]
    r <- theta[2,1]
    dev <- numericDeriv(quote((K*2)/(2+(K-2)*exp(-r*t))),theta.note)
    gradient.matrix <- matrix(attributes(dev)$gradient, nrow = 2)
    gradient.matrix0 <- rbind(gradient.matrix0, t(gradient.matrix))
  }

  a.matrix <- gradient.matrix0
  print(theta)
  print(sse)
  #Updating Matrix
  update <- solve(t(a.matrix) %*% a.matrix) %*% t(a.matrix) %*% z.matrix
  theta <- theta + update
}
```

```
##              [,1]
## [1,] 1200.0000000
## [2,]    0.1716788
## [1] 677146.3
##              [,1]
## [1,] 988.5766717
## [2,]   0.1367234
## [1] 110309.7
##               [,1]
## [1,] 1024.1437801
## [2,]    0.1224583
## [1] 76986.61
##               [,1]
## [1,] 1045.8468842
## [2,]    0.1189023
## [1] 73503.58
##               [,1]
## [1,] 1048.9806817
## [2,]    0.1183764
## [1] 73421.83
##               [,1]
## [1,] 1049.3390366
## [2,]    0.1182867
## [1] 73419.76
##               [,1]
## [1,] 1049.3958170
## [2,]    0.1182715
## [1] 73419.7
##               [,1]
## [1,] 1049.4053151
## [2,]    0.1182689
## [1] 73419.7
##               [,1]
## [1,] 1049.4069183
## [2,]    0.1182685
## [1] 73419.7
##               [,1]
## [1,] 1049.4071888
## [2,]    0.1182684
## [1] 73419.7
```

(b)

```r
#x <- as.numeric(1:5)


K <- seq(1049,1200, by = (1200-1049)/29)
r <- seq(0.1182,0.1716, by = (0.1716 - 0.1182)/29)

n.b <- as.numeric(beetles$beetles)
t.d <- as.numeric(beetles$days)

sse <- function(K,r,n,t){
  i <- 1
  error.sq <- 0
  for (i in 1:10){
  error.sq <- (n[i] - (K*2)/(2+(K-2)*exp(-r*t[i])))^2
  error.sq<- error.sq + error.sq
  }
  return(error.sq)
}

z <- matrix(rep(0,900),nrow = 30)
j <- 1 #for K
k <- 1 #for r
for (j in 1:30){
  for(k in 1:30){
    z[j,k] <- sse(K[j],r[k],n.b,t.d)
  }
}


filled.contour(K,r,z, xlab = 'K', ylab = 'r')
```



As shown in the contour plot, the sum of squared errors decrease while (K, r) approach (1049.4069185, 0.1182685) from the initial values $(K, r) = (1200, 0.1716788)$.

(c)

```
mlogl3 <- function(theta, N, days) {
  K <- theta[1]
  r <- theta[2]
  sigma <- theta[3]
  t <- days
  #mu <- log( f(...))
  mu <- log((K*2)/(2+(K-2)*exp(-r*t)))
  - sum(dnorm(log(N), mu, sigma, log = TRUE))
}

sqrt(var(log(beetles$beetles)))
```

```
## [1] 2.031806
```

```
theta.start <- c(1200, 0.17,2.03)
out <- nlm(mlogl3, theta.start, N=beetles$beetles,days=beetles$days,hessian = TRUE)
out
```

```
## $minimum
## [1] 9.790127
##
## $estimate
## [1] 820.3815695   0.1926394   0.6440836
##
## $gradient
## [1]  1.138721e-08  2.900435e-05 -2.714273e-06
##
## $hessian
##               [,1]         [,2]          [,3]
## [1,]  2.389705e-05   0.05442387 -3.048933e-06
## [2,]  5.442387e-02 373.52578204 -5.767475e-02
## [3,] -3.048933e-06  -0.05767475  4.817365e+01
##
## $code
## [1] 2
##
## $iterations
## [1] 38
```

```
theta.hat <- out$estimate
#K = 820.3811422 , r = 0.1926394, sigma = 0.6440836
theta.hat
```

```
## [1] 820.3815695   0.1926394   0.6440836
```

```
hes <- out$hessian
hes
```

```
##               [,1]         [,2]          [,3]
## [1,]  2.389705e-05   0.05442387 -3.048933e-06
## [2,]  5.442387e-02 373.52578204 -5.767475e-02
## [3,] -3.048933e-06  -0.05767475  4.817365e+01
```

```
var.matrix <- solve(hes)
# 6.262790e+04, 4.006745e-03, 2.075824e-02
diag(var.matrix)
```

```
## [1] 6.262790e+04 4.006745e-03 2.075824e-02
```