

# STAT 5361 Statistical Computing Solution

*2018-10-03*



# Contents

<b>1</b>	<b>Exercise 3.1</b>	<b>5</b>
1.1	Get Fisher Information . . . . .	5
1.2	Implement loglikelihood with a random sample and plot against $\theta$ . . . . .	6
1.3	Newton-Raphson method . . . . .	7
1.4	Fixed point method . . . . .	8
1.5	Fisher scoring and Newton-Raphson . . . . .	9
1.6	comparing the different methods . . . . .	10
<b>2</b>	<b>Exercise 3.2</b>	<b>13</b>
2.1	Find the the log-likelihood function . . . . .	13
2.2	Find the method-of-moments estimator . . . . .	14
2.3	Find the MLE . . . . .	14
2.4	$\theta_0 = 2.7$ or $\theta_0 = -2.7$ . . . . .	15
2.5	Repeat the above using 200 equally spaced starting values . . . . .	16
<b>3</b>	<b>Exercise 3.3</b>	<b>21</b>
3.1	Fit the population growth model to the beetles data using the Gauss-Newton approach . . .	21
3.2	Log normal assumption . . . . .	23



# Chapter 1

## Exercise 3.1

### 1.1 Get Fisher Information

$$\begin{aligned}f(x; \theta) &= \frac{1}{\pi(1 + (x - \theta)^2)} \\ \Rightarrow l(\theta) &= \sum_{i=1}^n \ln(f(X_i; \theta)) = -n \ln(\pi) - \sum_{i=1}^n \ln(1 + (X_i - \theta)^2) \\ \Rightarrow l'(\theta) &= -2 \sum_{i=1}^n \frac{\theta - X_i}{1 + (\theta - X_i)^2} \\ \Rightarrow l''(\theta) &= -2 \sum_{i=1}^n \left[ \frac{1}{1 + (\theta - X_i)^2} - \frac{2(\theta - X_i)^2}{[1 + (\theta - X_i)^2]^2} \right] = -2 \sum_{i=1}^n \frac{1 - (\theta - X_i)^2}{[1 + (\theta - X_i)^2]^2} \\ \Rightarrow I_n(\theta) &= -El''(\theta) \\ &= 2nE \frac{1 - (\theta - X)^2}{[1 + (\theta - X)^2]^2} \\ &= \frac{2n}{\pi} \int_R \frac{1 - (x - \theta)^2}{(1 + (x - \theta)^2)^3} dx \\ &= \frac{2n}{\pi} \int_R \frac{1 - x^2}{(1 + x^2)^3} dx \\ &= \frac{2n}{\pi} \int_R \frac{-1}{(1 + x^2)^2} + 2 \frac{2}{(1 + x^2)^3} dx\end{aligned}$$

Also:

$$\begin{aligned}
M_k &= \int_R \frac{1}{(1+x^2)^k} dx \\
&= \int_R \frac{1+x^2}{(1+x^2)^{k+1}} dx \\
&= M_{k+1} + \int_R \frac{x^2}{(1+x^2)^{k+1}} dx \\
&= M_{k+1} + \int_R \frac{2kx}{(1+x^2)^{k+1}} \frac{x}{2k} dx = M_{k+1} + \frac{1}{2k} M_k
\end{aligned}$$

Since  $M_1 = \pi$ , we have  $M_2 = \pi/2$ ,  $M_3 = 3\pi/8$ , then  $I_n(\theta) = n/2$ .

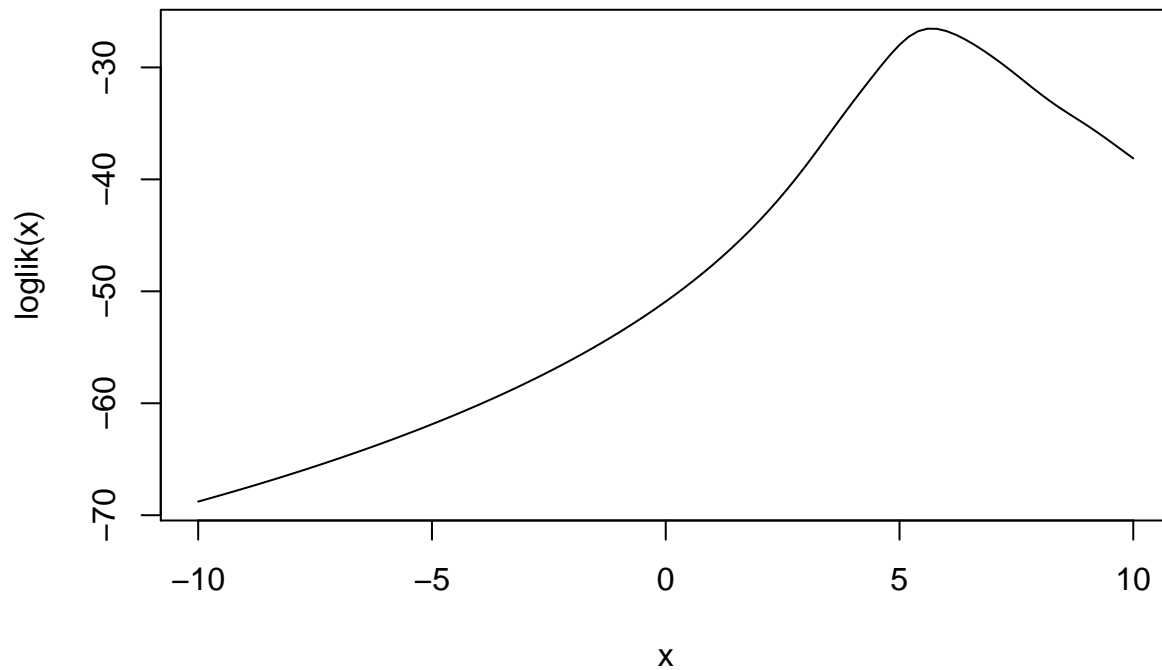
## 1.2 Implement loglikelihood with a random sample and plot against $\theta$

Use the loglikelihood function we got from above, set  $n = 10$  and plug in the generated sample value  $X_i$ , we can get the loglikelihood function. When generating sample, the location parameter was set to be  $\theta = 5$ . The loglikelihood function curve against  $\theta$  are shown in Figure ??:

```

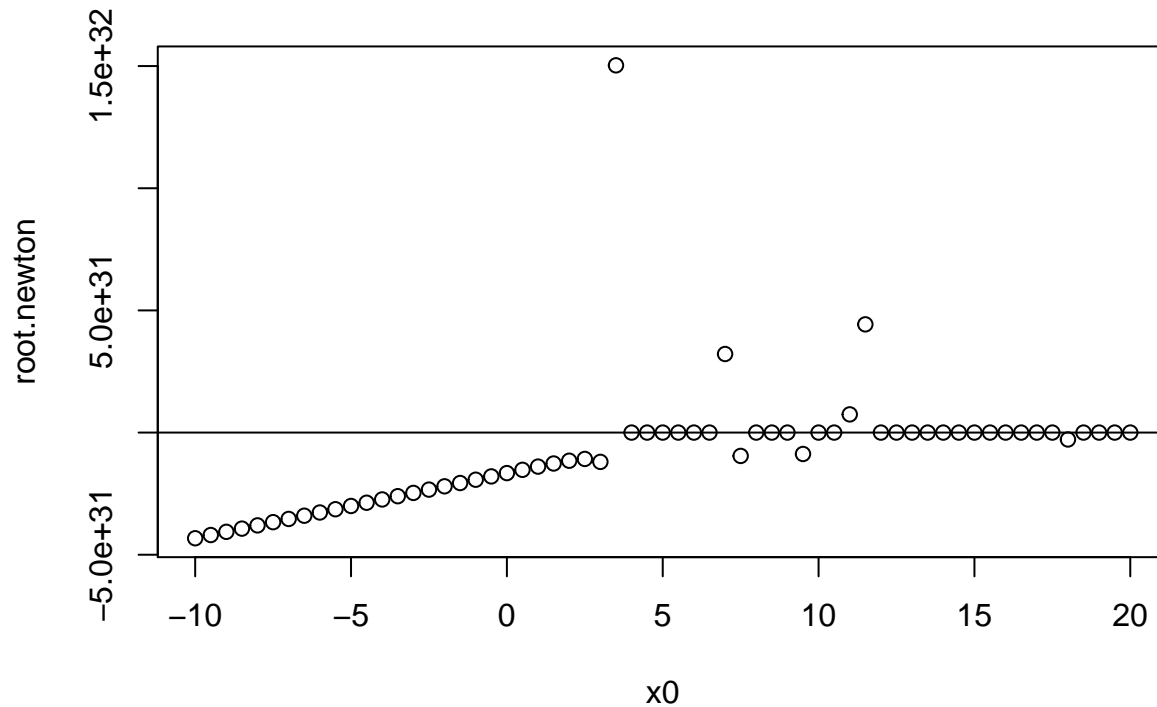
set.seed(20180909)
n <- 10
X <- rcauchy(n, location = 5, scale = 1)
loglik.0 <- function(theta) {
  l <- sum(dcauchy(X, location = theta, log = TRUE))
  l
}
loglik <- function(theta) {
  l <- sapply(theta, FUN = loglik.0)
  l
}
curve(loglik, -10, 10)

```



### 1.3 Newton-Raphson method

```
library(pracma)
## define the derivative function
dev.loglik <- function(theta) {
  dev.l <- -2 * sum((theta-X)/(1+(theta-X)^2))
  dev.l
}
## define the hessian function
hessian.loglik <- function(theta) {
  h <- -2 * sum((1-(theta-X)^2) * (1+(theta-X)^2)^(-2))
  h
}
x0 <- seq(-10, 20, by = 0.5)
root.newton <- rep(0, length(x0))
for (i in 1:length(x0)) {
  root.newton[i] <- newtonRaphson(dev.loglik, x0 = x0[i], dfun = hessian.loglik)$root
}
plot(x0, root.newton)
abline(h = 5.442)
```



```
root.newton
```

```
## [1] -4.324741e+31 -4.193577e+31 -4.062249e+31 -3.930748e+31 -3.799064e+31
## [6] -3.667185e+31 -3.535100e+31 -3.402796e+31 -3.270261e+31 -3.137479e+31
## [11] -3.004436e+31 -2.871118e+31 -2.737510e+31 -2.603599e+31 -2.469374e+31
## [16] -2.334832e+31 -2.199981e+31 -2.064850e+31 -1.929508e+31 -1.794100e+31
## [21] -1.658922e+31 -1.524582e+31 -1.392396e+31 -1.265439e+31 -1.151924e+31
## [26] -1.079358e+31 -1.199750e+31 1.502957e+32 2.056366e+01 2.108229e+01
## [31] 5.685422e+00 5.685422e+00 5.685422e+00 5.685422e+00 3.215974e+31
## [36] -9.558888e+30 1.937744e+01 2.108229e+01 5.685422e+00 -8.759488e+30
## [41] 2.108229e+01 5.685422e+00 7.439560e+30 4.429077e+31 2.056366e+01
## [46] 2.056366e+01 2.056366e+01 2.108229e+01 2.108229e+01 2.108230e+01
## [51] 1.937743e+01 2.056366e+01 2.056366e+01 1.937744e+01 1.937743e+01
## [56] 1.937744e+01 -2.825479e+30 2.056366e+01 2.056366e+01 2.056366e+01
## [61] 2.056366e+01
```

We can see that Newton method doesn't converge when initial value is not close to the real root.

## 1.4 Fixed point method

```
## self-defined fixed point methods to find mle
## input gradient of loglikelihood function, x0 is initial value
FixPoint.mle <- function(dev.loglik, alpha, x0, maxiter = 100,
                        tol = .Machine$double.eps^0.5){
  x <- x0
  for (i in 1:maxiter) {
    x.new <- alpha * dev.loglik(x) + x
    if (abs(x.new - x) < tol) break
    x <- x.new
  }
}
```

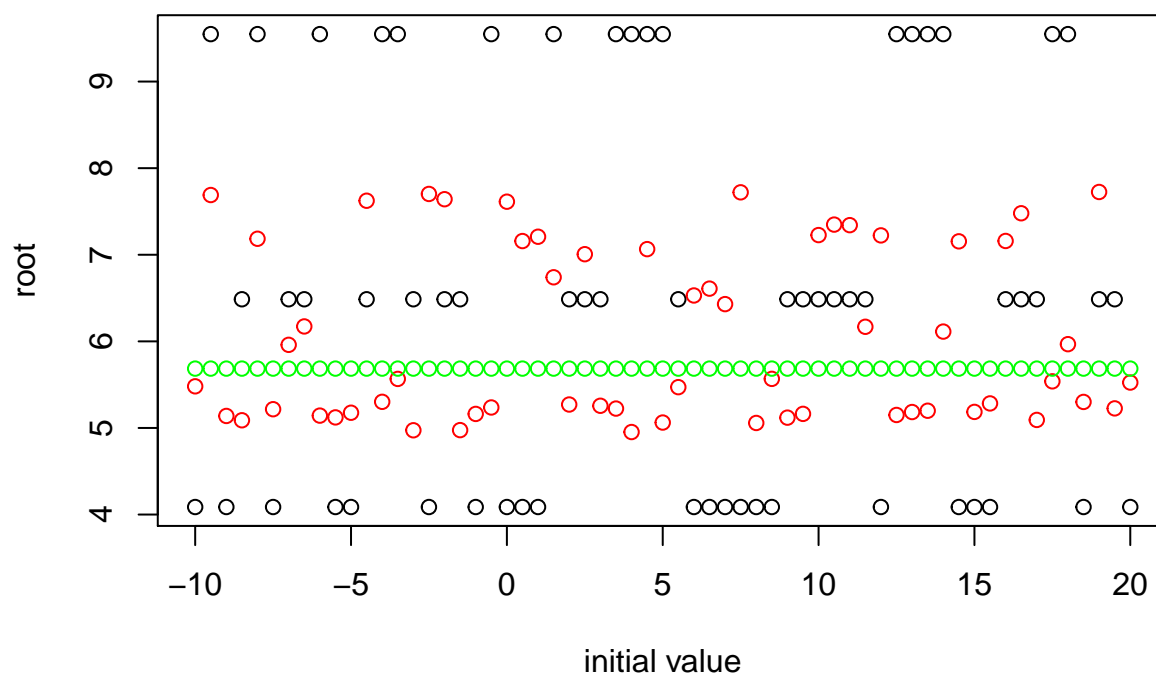


```

    if (i == maxiter) warning("maximum iteration has reached")
    return(list(root = x, niter = i))
}
alpha <- c(1, 0.64, 0.25)
root.fixpoint <- matrix(0, ncol = length(alpha), nrow = length(x0))
for (i in 1:length(alpha)) {
  for (j in 1:length(x0)) {
    root.fixpoint[j, i] <- FixPoint.mle(dev.loglik = dev.loglik, alpha = alpha[i],
                                         x0 = x0[j])$root
  }
}
plot(x0, root.fixpoint[, 1], ylim = c(min(root.fixpoint), max(root.fixpoint)),
     ylab = "root", xlab = "initial value",
     main = paste0("black: ", expression(alpha), "= 1; red: ", expression(alpha),
                   "= 0.64; green: ", expression(alpha), "= 0.25"))
points(x0, root.fixpoint[, 2], col = "red")
points(x0, root.fixpoint[, 3], col = "green")

```

**black: alpha= 1; red: alpha= 0.64; green: alpha= 0.25**



## 1.5 Fisher scoring and Newton-Raphson

```

## Self-defined fisher scoring method to find mle.
## input gradient of loglikelihood and sample fisher information.
FisherScore.mle <- function(dev.loglik, information, x0, maxiter = 100,
                             tol = .Machine$double.eps^0.5) {
  x <- x0
  for (i in 1:maxiter) {
    x.new <- x + dev.loglik(x) / information(x)
  }
}

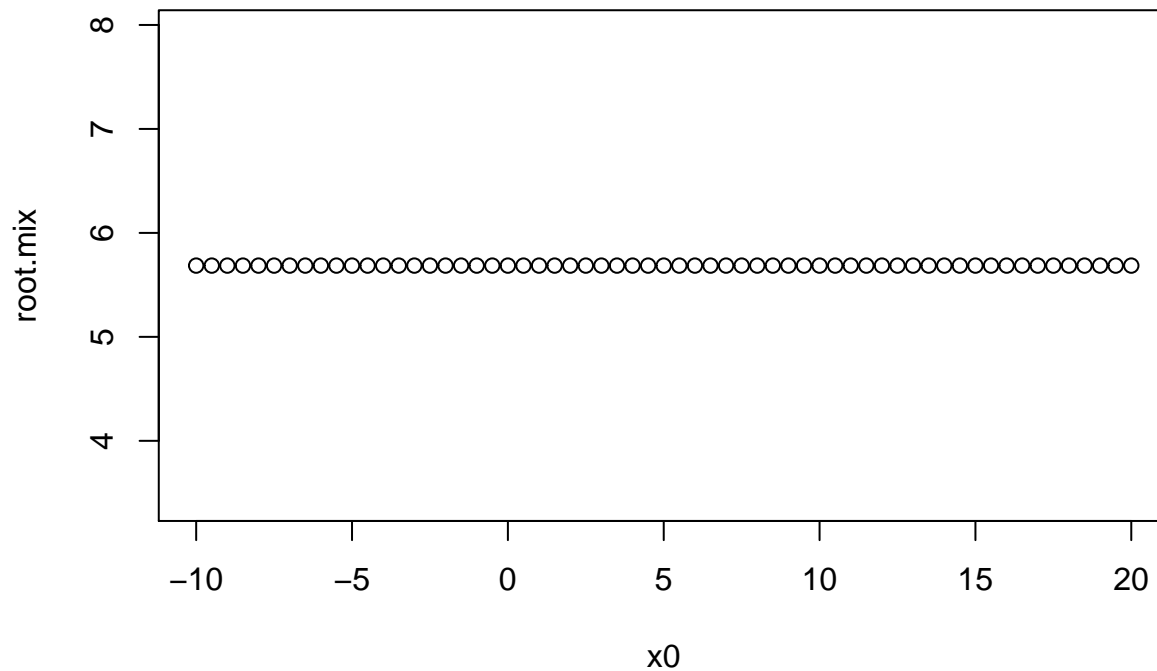
```

```

    if (abs(x.new - x) < tol) break
    x <- x.new
  }
  if (i == maxiter) warning("maximum iteration has reached")
  return(list(root = x, niter = i))
}
FisherNewton.mle <- function(dev.loglik, information, dfun, x0, maxiter = 100,
                             tol = .Machine$double.eps^0.5) {
  method.fisher <- FisherScore.mle(dev.loglik = dev.loglik, information = information,
                                   x0 = x0, maxiter = maxiter, tol = tol)
  x.fisher <- method.fisher$root
  niter.fisher <- method.fisher$niter
  method.newton <- newtonRaphson(fun = dev.loglik, x0 = x.fisher, dfun = dfun, maxiter = maxiter,
                                tol = tol)
  return(list(root = method.newton$root, niter.fisher = niter.fisher,
              niter.newton = method.newton$niter))
}
inf.cauchy <- function(x) n/2

root.mix <- rep(0, length(x0))
for (i in 1:length(x0)) {
  root.mix[i] <- FisherNewton.mle(dev.loglik, inf.cauchy, dfun = hessian.loglik,
                                  x0 = x0[i])$root
}
plot(x0, root.mix)

```



## 1.6 comparing the different methods

```

library(microbenchmark)
## comparing the speed of different methods

```

```
## use starting point 5, alpha = 0.25 for fixed point method
newton.method <- newtonRaphson(fun = dev.loglik, x0 = 5, dfun = hessian.loglik)
fixpoint.method <- FixPoint.mle(dev.loglik = dev.loglik, alpha = 0.25, x0 = 5)
fishernewton.method <- FisherNewton.mle(dev.loglik = dev.loglik, information = inf.cauchy,
                                         dfun = hessian.loglik, x0 = 5)
list(newton.niter = newton.method$niter, fixpoint.niter = fixpoint.method$niter,
     fishernewton.niter = c(fishernewton.method$niter.fisher,
                           fishernewton.method$niter.newton))
```

```
## $newton.niter
## [1] 6
##
## $fixpoint.niter
## [1] 17
##
## $fishernewton.niter
## [1] 8 1
```

Fixed point method is most stable but converges slowly compare to the other two methods. Newton-Raphson methods converges fastest but is the most unstably one. Fisher-Scoring converges slower than Newton, but is very stable and accuracy, after refining with Newton-Raphson methods. Also we can see that if we use fisher scoring root to be the initial value of Newton-Raphson method, it will converge very fast.



## Chapter 2

### Exercise 3.2

#### 2.1 Find the the log-likelihood function

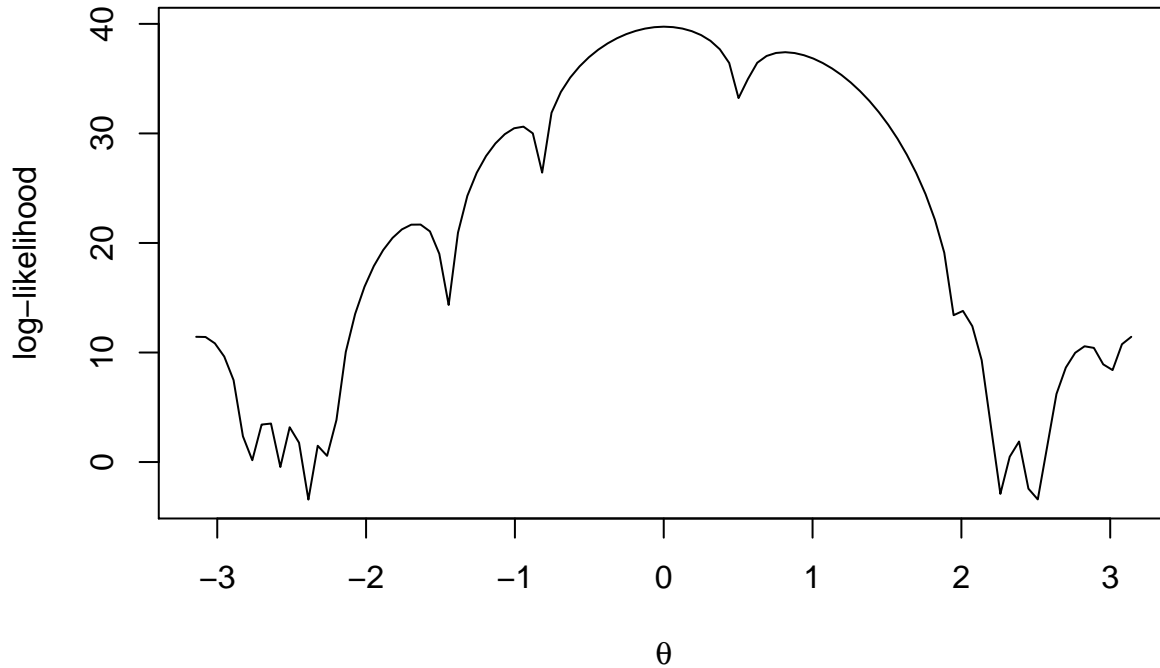
The log-likelihood function of this distribution is

$$\ell(\mathbf{x}, \theta) = \sum_{i=1}^n \log\{1 - \cos(x_i - \theta)\} - n \log 2\pi$$

```
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)

loglikelihood <- function(theta){
  n <- length(x)
  s <- sum(log(1 - cos(x - theta))) + n * log(2 * pi)
  return(s)
}

loglikelihood <- Vectorize(loglikelihood)
curve(loglikelihood, -pi, pi, xlab = expression(theta), ylab = "log-likelihood")
```



## 2.2 Find the method-of-moments estimator

The expectation of  $\mathbf{x}|\theta$  is

$$\begin{aligned}
 \mathbb{E}(x|\theta) &= \int_0^{2\pi} x \frac{1 - \cos(x - \theta)}{2\pi} dx \\
 &= \frac{1}{2\pi} \int_0^{2\pi} x - x \cos(x - \theta) dx \\
 &= \pi + \sin(\theta) \\
 &= \bar{X}_n
 \end{aligned}$$

Thus,

```
theta_tilde <- asin(mean(x) - pi)
theta_tilde
```

```
## [1] 0.09539407
```

## 2.3 Find the MLE

Since

$$\begin{aligned}
 \frac{\partial \ell(\mathbf{x}; \theta)}{\partial \theta} &= \sum_{i=1}^n \frac{-\sin(x_i - \theta)}{1 - \cos(x_i - \theta)} \\
 \frac{\partial^2 \ell(\mathbf{x}; \theta)}{\partial \theta^2} &= \sum_{i=1}^n \frac{\cos(x_i - \theta) - \cos^2(x_i - \theta) - \sin^2(x_i - \theta)}{(1 - \cos(x_i - \theta))^2}
 \end{aligned}$$

The Newton-Raphson method is

$$\hat{\theta}^{(t+1)} = \hat{\theta}^{(t)} - \left\{ \frac{\partial^2 \ell(\mathbf{x}; \hat{\theta}^{(t)})}{\partial \theta^2} \right\}^{-1} \frac{\partial \ell(\mathbf{x}; \hat{\theta}^{(t)})}{\partial \theta}$$

```
lfd <- function(theta){
  sum(-sin(x-theta)/(1-cos(x-theta)))
}

lsd <- function(theta){
  sum((cos(x-theta) - (cos(x-theta))^2 - (sin(x-theta))^2)/(1-cos(x-theta))^2)
}

Newton <- function(init){
  theta0 <- init
  i <- 0
  diff <- 1
  msg <- "converge"
  while(abs(diff) > 0.0000001){
    lfd <- lfd(theta0)
    lsd <- lsd(theta0)
    diff <- (lfd/lsd)
    theta1 <- theta0 - diff
    theta0 <- theta1
    i <- i+1
    #cat(i)
    if(i >= 150){
      msg <- "Not converge"
      theta0 <- Inf
      break
    }
  }
  return(list(theta = theta0, itr = i, msg = msg))
}

Newton(theta_tilde)

## $theta
## [1] 0.003118157
##
## $itr
## [1] 4
##
## $msg
## [1] "converge"
```

## 2.4 $\theta_0 = 2.7$ or $\theta_0 = -2.7$

```
Newton(-2.7)
```

```
## $theta
## [1] -2.668857
##
```

```
## $itr
## [1] 4
##
## $msg
## [1] "converge"
```

```
Newton(2.7)
```

```
## $theta
## [1] 2.848415
##
## $itr
## [1] 5
##
## $msg
## [1] "converge"
```

The  $\hat{\theta}$  we got is different.

## 2.5 Repeat the above using 200 equally spaced starting values

```
init <- seq(-pi, pi, length.out=200)
result <- NULL
for(initi in init){
  result <- rbind(result, c(initi, Newton(initi)$theta))
}
colnames(result) <- c("Initial_value", "theta_hat")
split(result, result[,2])
```

```
## $^-3.11247050669846`
## [1] -3.141593 -3.110019 -3.078445 -3.046871 -3.015297 -2.983724 -2.952150
## [8] -2.920576 -2.889002 -2.857428 -2.825855 -3.112471 -3.112471 -3.112471
## [15] -3.112471 -3.112471 -3.112471 -3.112471 -3.112471 -3.112471 -3.112471
## [22] -3.112471
##
## $^-2.78655685241805`
## [1] -2.794281 -2.786557
##
## $^-2.78655685241804`
## [1] -2.762707 -2.786557
##
## $^-2.66885745902142`
## [1] -2.731133 -2.699560 -2.667986 -2.636412 -2.604838 -2.668857 -2.668857
## [8] -2.668857 -2.668857 -2.668857
##
## $^-2.50935603320277`
## [1] -2.573264 -2.541691 -2.510117 -2.478543 -2.446969 -2.415395 -2.509356
## [8] -2.509356 -2.509356 -2.509356 -2.509356 -2.509356
##
## $^-2.38826662826452`
## [1] -2.383822 -2.388267
##
## $^-2.29792596896698`
```



```

## [1] -2.352248 -2.297926
##
## $^-2.29792596896697`
## [1] -2.320674 -2.289100 -2.257526 -2.297926 -2.297926 -2.297926
##
## $^-2.23219189887219`
## [1] -2.225953 -2.232192
##
## $^-1.66271239546243`
## [1] -2.194379 -2.162805 -2.131231 -2.099657 -2.068084 -2.036510 -2.004936
## [8] -1.973362 -1.941788 -1.910215 -1.878641 -1.847067 -1.815493 -1.783919
## [15] -1.752346 -1.720772 -1.689198 -1.594477 -1.531329 -1.499755 -1.468181
## [22] -1.662712 -1.662712 -1.662712 -1.662712 -1.662712 -1.662712 -1.662712
## [29] -1.662712 -1.662712 -1.662712 -1.662712 -1.662712 -1.662712 -1.662712
## [36] -1.662712 -1.662712 -1.662712 -1.662712 -1.662712 -1.662712 -1.662712
##
## $^-1.66271239546242`
## [1] -1.657624 -1.626050 -1.562903 -1.662712 -1.662712 -1.662712
##
## $^-1.44750255268373`
## [1] -1.436608 -1.447503
##
## $^-0.95440583712848`
## [1] -1.4050339 -1.3103125 -1.2787387 -1.2155911 -1.1208697 -1.0577221
## [7] -1.0261484 -0.9945746 -0.9544058 -0.9544058 -0.9544058 -0.9544058
## [13] -0.9544058 -0.9544058 -0.9544058 -0.9544058
##
## $^-0.954405837128479`
## [1] -1.3734601 -1.3418863 -1.1524435 -1.0892959 -0.9630008 -0.8998532
## [7] -0.8682794 -0.8367056 -0.9544058 -0.9544058 -0.9544058 -0.9544058
## [13] -0.9544058 -0.9544058 -0.9544058 -0.9544058
##
## $^-0.954405837128476`
## [1] -0.9314270 -0.9544058
##
## $^-0.95440583712847`
## [1] -1.2471649 -0.9544058
##
## $^-0.954405837128466`
## [1] -1.1840173 -0.9544058
##
## $^0.00311815708656577`
## [1] -0.489393830 0.003118157
##
## $^0.0031181570865658`
## [1] -0.078934489 0.003118157
##
## $^0.00311815708656581`
## [1] 0.110508284 0.003118157
##
## $^0.00311815708656585`
## [1] -0.236803466 0.003118157
##
## $^0.00311815708656587`

```

```

## [1] -0.142082080 -0.047360693 0.003118157 0.003118157
##
## $`0.00311815708656589`
## [1] -0.678836604 -0.584115217 0.003118157 0.003118157
##
## $`0.00311815708656591`
## [1] -0.110508284 0.015786898 0.003118157 0.003118157
##
## $`0.00311815708656593`
## [1] -0.710410399 0.142082080 0.299951057 0.003118157 0.003118157
## [6] 0.003118157
##
## $`0.00311815708656597`
## [1] -0.457820035 0.003118157
##
## $`0.00311815708656598`
## [1] -0.015786898 0.236803466 0.268377262 0.003118157 0.003118157
## [6] 0.003118157
##
## $`0.00311815708656599`
## [1] -0.805131786 0.003118157
##
## $`0.003118157086566`
## [1] -0.741984195 0.003118157
##
## $`0.00311815708656601`
## [1] -0.268377262 0.394672444 0.003118157 0.003118157
##
## $`0.00311815708656602`
## [1] -0.647262808 -0.552541421 0.078934489 0.003118157 0.003118157
## [6] 0.003118157
##
## $`0.00311815708656603`
## [1] -0.773557990 -0.615689013 0.047360693 0.489393830 0.003118157
## [6] 0.003118157 0.003118157 0.003118157
##
## $`0.00311815708656604`
## [1] -0.363098648 0.003118157
##
## $`0.00311815708656606`
## [1] 0.457820035 0.003118157
##
## $`0.00311815708656607`
## [1] -0.426246239 0.003118157
##
## $`0.00311815708656609`
## [1] -0.173655875 0.003118157
##
## $`0.00311815708656611`
## [1] -0.205229671 0.003118157
##
## $`0.00311815708656612`
## [1] -0.394672444 0.363098648 0.003118157 0.003118157
##

```

```

## $`0.00311815708656613`
## [1] -0.299951057 0.003118157
##
## $`0.00311815708656615`
## [1] 0.205229671 0.003118157
##
## $`0.00311815708656793`
## [1] 0.426246239 0.003118157
##
## $`0.00311815708656861`
## [1] -0.520967626 0.003118157
##
## $`0.00311815708656864`
## [1] 0.331524853 0.003118157
##
## $`0.00311815708656926`
## [1] -0.331524853 0.003118157
##
## $`0.00311815708656987`
## [1] 0.173655875 0.003118157
##
## $`0.812637416717926`
## [1] 1.2787387 0.8126374
##
## $`0.812637416717938`
## [1] 0.8051318 1.4050339 0.8126374 0.8126374
##
## $`0.812637416717939`
## [1] 0.6788366 0.8126374
##
## $`0.81263741671794`
## [1] 0.5209676 0.5525414 0.5841152 0.6156890 0.6472628 0.7104104 0.7419842
## [8] 0.7735580 0.8367056 0.8682794 0.8998532 0.9314270 0.9630008 0.9945746
## [15] 1.0261484 1.0577221 1.0892959 1.1208697 1.1524435 1.1840173 1.2155911
## [22] 1.2471649 1.3103125 1.3418863 1.3734601 1.4366077 1.4681815 1.4997553
## [29] 1.5313291 1.5629029 1.5944767 1.6260505 1.6576243 1.6891981 1.7207719
## [36] 1.7523457 1.7839194 1.8154932 1.8470670 1.8786408 1.9102146 1.9417884
## [43] 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374
## [50] 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374
## [57] 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374
## [64] 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374
## [71] 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374
## [78] 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374 0.8126374
##
## $`2.00722323801594`
## [1] 1.973362 2.004936 2.068084 2.099657 2.131231 2.162805 2.194379
## [8] 2.007223 2.007223 2.007223 2.007223 2.007223 2.007223 2.007223
##
## $`2.00722323801595`
## [1] 2.036510 2.007223
##
## $`2.23701292270577`
## [1] 2.225953 2.257526 2.237013 2.237013
##

```

```
## $`2.37471166606864`  
## [1] 2.289100 2.320674 2.352248 2.383822 2.415395 2.446969 2.374712  
## [8] 2.374712 2.374712 2.374712 2.374712 2.374712  
##  
## $`2.48844965088485`  
## [1] 2.478543 2.488450  
##  
## $`2.48844965088489`  
## [1] 2.510117 2.488450  
##  
## $`2.84841532545741`  
## [1] 2.541691 2.573264 2.604838 2.636412 2.667986 2.699560 2.731133  
## [8] 2.762707 2.794281 2.825855 2.857428 2.920576 2.952150 2.983724  
## [15] 2.848415 2.848415 2.848415 2.848415 2.848415 2.848415 2.848415  
## [22] 2.848415 2.848415 2.848415 2.848415 2.848415 2.848415 2.848415  
##  
## $`2.84841532545742`  
## [1] 2.889002 2.848415  
##  
## $`3.17071480048113`  
## [1] 3.015297 3.046871 3.078445 3.110019 3.141593 3.170715 3.170715  
## [8] 3.170715 3.170715 3.170715
```

## Chapter 3

### Exercise 3.3

#### 3.1 Fit the population growth model to the beetles data using the Gauss-Newton approach

```
library(graphics)
library(Matrix)
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))

##' define the sum of squared errors function
sqerr <- function(k, r) {
  s <- matrix(0, nrow = length(k), ncol = length(r))
  for (i in 1:length(k)) {
    for (j in 1:length(r)) {
      s[i, j] <- sum((beetles$beetles - k[i] * beetles$beetles[1] /
        (beetles$beetles[1] + (k[i] - beetles$beetles[1]) *
          exp(-r[j]*beetles$days)))^2)
    }
  }
  s
}

##' define z function
z.vec <- function(k, r) {
  n <- length(beetles$days)
  z <- rep(0, n)
  for (i in 1:n) {
    z[i] <- beetles$beetles[i] - k*beetles$beetles[1] /
      (beetles$beetles[1] + (k - beetles$beetles[1])*exp(-r*beetles$days[i]))
  }
  return(z)
}

##' define A matrix
A.mat <- function(k, r) {
  n <- length(beetles$days)
```

```

A <- matrix(0, nrow = n, ncol = 2)
for (i in 1:n) {
  A[i, 1] <- beetles$beetles[1]^2 * (1-exp(-r*beetles$days[i])) /
    (beetles$beetles[1] + (k-beetles$beetles[1])*exp(-r*beetles$days[i]))^2

  A[i, 2] <- beetles$beetles[1]*k*beetles$days[i]*(k-beetles$beetles[1]) *
    exp(-r*beetles$days[i]) / (beetles$beetles[1]+(k-beetles$beetles[1]) *
      exp(-r*beetles$days[i]))^2
}
return(A)
}

gaussNewton.beetles <- function(para0, z.vec, A.mat, maxiter = 100,
                                tol = .Machine$double.eps^0.5) {
  para <- para0
  for (i in 1:maxiter) {
    Amat <- A.mat(para[1], para[2])
    zvec <- z.vec(para[1], para[2])
    para.new <- para + solve(t(Amat) %*% Amat + 0.0001*diag(nrow = 2)) %*%
      t(Amat) %*% zvec
    if (sum(abs(para.new - para)) < tol) break
    para <- para.new
  }
  if (i == maxiter) warning("maximum iteration has reached")
  return(list(root = para, niter = i))
}

fit <- gaussNewton.beetles(para0 = c(1200, 0.1), z.vec = z.vec, A.mat = A.mat)
fit$root

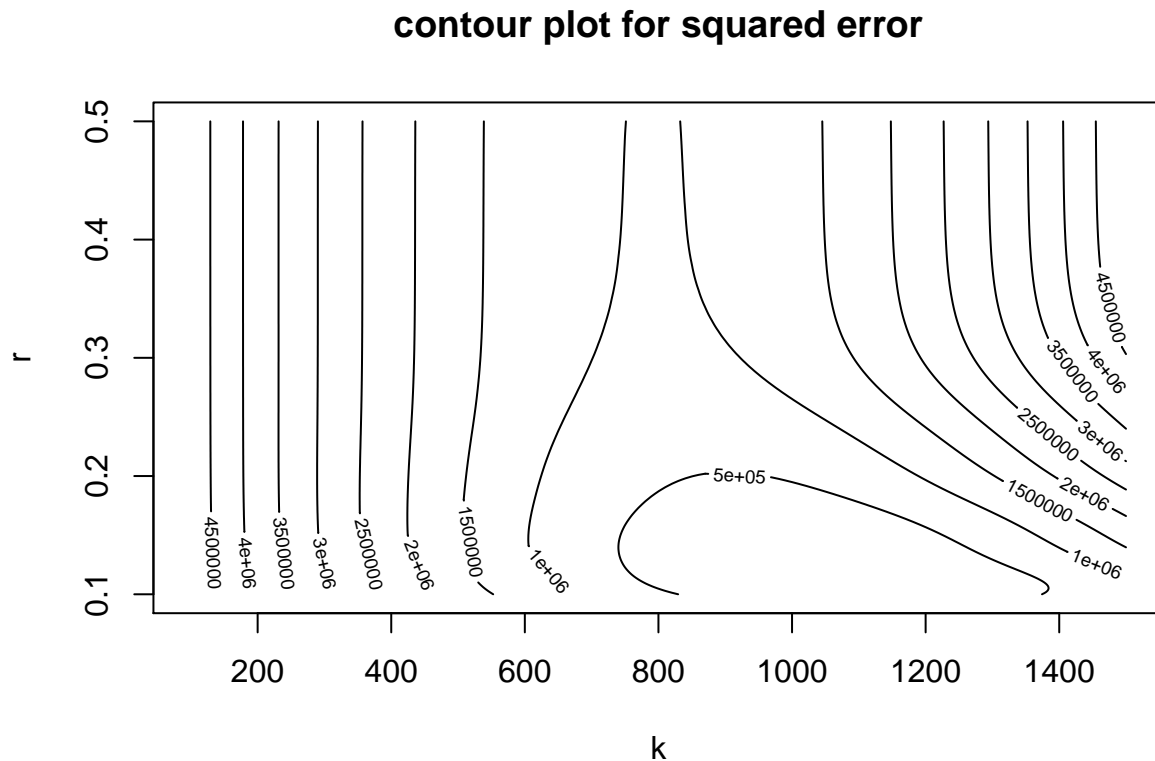
```

```

##           [,1]
## [1,] 1049.4072443
## [2,]    0.1182684

k <- seq(100, 1500, by = 10)
r <- seq(0.1, 0.5, by = 0.001)
z <- sqerr(k, r)
contour(k, r, z, xlab = "k", ylab = "r", main = "contour plot for squared error")

```



Estimation using Gauss-Newton method is  $\hat{k} = 1049$ ,  $\hat{r} = 0.12$ .

## 3.2 Log normal assumption

```
##' define log-likelihood function
loglikeli <- function(para) {
  k <- para[1]
  r <- para[2]
  sigma2 <- para[3]
  l <- sum(-log(2*pi*sigma2)/2 - (log(beetles$beetles) - log(k) -
    log(beetles$beetles[1]) +
    log(beetles$beetles[1] +
      (k-beetles$beetles[1]) *
      exp(-r*beetles$days))^2)/2/sigma2)

  return(l)
}

##' define gradient of loglikelihood function
grad.my <- function(para) {
  k <- para[1]
  r <- para[2]
  sigma2 <- para[3]
  g <- rep(0, 3)
  g[1] <- sum(-2*(log(beetles$beetles)-log(k)-log(beetles$beetles[1]))+
    log(beetles$beetles[1]+(k-beetles$beetles[1])*
      exp(-r*beetles$days)))*
    (-1/k+exp(-r*beetles$days)/
      (beetles$beetles[1] +
```

```

      (k-beetles$beetles[1])*exp(-r*beetles$days)))/2/sigma2)
g[2] <- sum(2*(log(beetles$beetles)-log(k)-log(beetles$beetles[1])+
      log(beetles$beetles[1]+(k-beetles$beetles[1])*
      exp(-r*beetles$days))*
      beetles$days*(k-beetles$beetles[1])*exp(-r*beetles$days)/
      (beetles$beetles[1]+(k-beetles$beetles[1])*exp(-r*beetles$days))/
      2/sigma2)
g[3] <- sum(-1/2/sigma2+(log(beetles$beetles)-log(k)-log(beetles$beetles[1])+
      log(beetles$beetles[1]+(k-beetles$beetles[1])*
      exp(-r*beetles$days)))/2/2/sigma2^2)

  return(g)
}

fit <- constrOptim(theta = c(10, 0.1, 1), f = loglikeli, grad = grad.my,
  ui = diag(1, 3), ci = rep(0, 3),
  control = list(fnscale = -1), hessian = TRUE)

fit$par

## [1] 103.983292 12.124656 2.913935

fit$convergence

## [1] 0

Diag(-solve(fit$hessian))

## [1] 2.707176e+03 1.212466e+05 2.841633e+00

```

Using BFGS methods, set constrain to make  $k$ ,  $r$ ,  $\sigma^2$  nonnegative, we get the MLE estimates  $\hat{k} = 103.98$ ,  $\hat{r} = 12.12$ ,  $\hat{\sigma}^2 = 2.91$ . Using inverse of negative hessian, we have these estimates' variance to be  $2.7 \times 10^3$ ,  $1.2 \times 10^5$ , 2.8 respectively.

But there's problem that results will change dramatically with initial values. I just tried several different initial values, get the estimates and compare their corresponding loglikelihood. I just choose the one with maximum loglikelihood.