Statistical Computing Homework 4, Chapter 3

Ziqi Yang

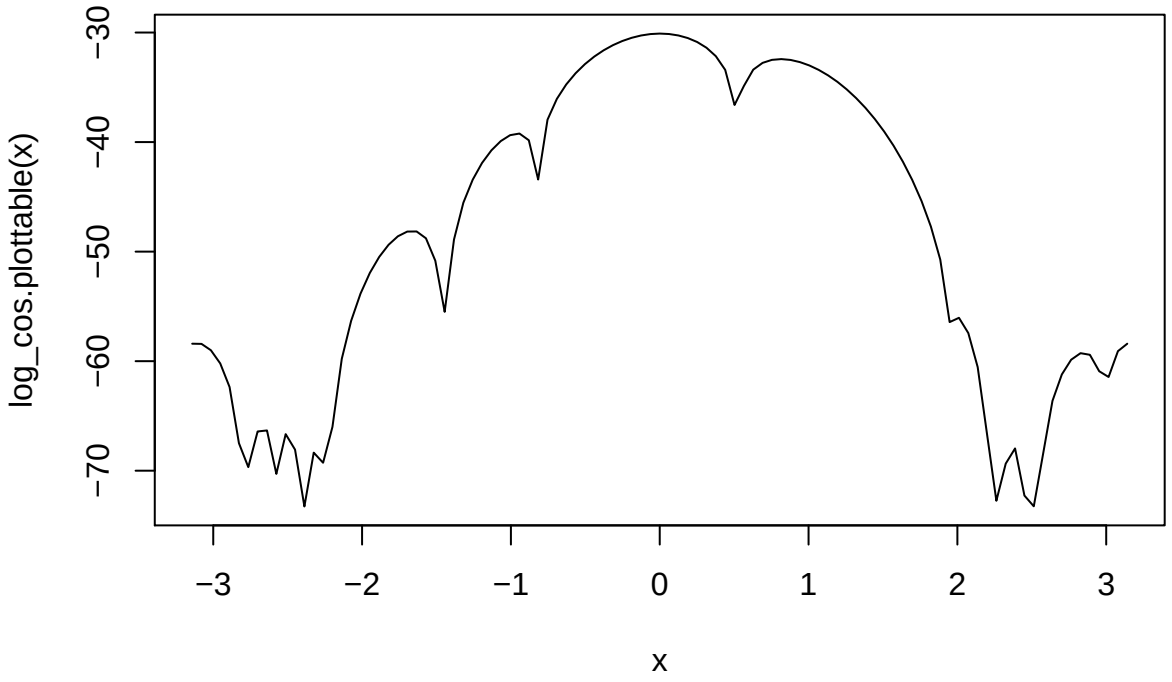
28 September, 2018

Contents

3.3.2 Many local maxima

(a) log-likelihood

```
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)
log_cos <- function(theta) {
  sum( log(1 - cos(x - theta)) ) - length(x)*log(2*pi)
}
log_cos.plottable <- function(x) {return(sapply(x, log_cos))}
curve(log_cos.plottable, from = -pi, to = pi)
```



(b) Methods of moment

Let

$$\frac{1}{2\pi}\int_0^{2\pi}\big[x - x\cos(x-\theta)\big]dx = \bar{X}$$

so evetually, we have $\pi + \sin(\theta) = \bar{X}$, so MOM $\tilde{\theta}_n = \arcsin(\bar{X} - \pi) = 0.0953941$

(c) Find MLE using Newton-Raphson

```
log_cos_D1 <- function(theta) {
  sum( sin(theta-x)/(1-cos(x-theta)) )
}

log_cos_D2 <- function(theta) {
  sum( 1+cos(theta-x)/(1-cos(theta-x))^2 )
}

para <- c(-2.7, 2.7);  temp <- rep(0, length(para))
for (i in 1:length(para)) {
  iter <- 0;  epsilon <- 0.001;
  temp[i] <- para[i] - 1
  while( (iter <= 1000)&(abs(para[i]-temp[i])/(abs(temp[i])) > epsilon) ) {
    temp[i] <- para[i]
    para[i] <- para[i] - log_cos_D1(para[i])/log_cos_D2(para[i])
    if(abs(para[i]) == Inf) {break}
    iter <- iter + 1
    #print(para[i])
  }
}
print(para)
```

```
## [1] -2.700049  2.697756
```

We can see that if we start from -2.7 and 2.7, the Newton_Raphson algorithm will give us the similar result with the initial values.

(d)

```
para <- seq(-pi, pi, length.out = 200);  temp <- rep(0, length(para))
for (i in 1:length(para)) {
  iter <- 0;  epsilon <- 0.001;
  temp[i] <- para[i] - 1
  while( (iter <= 1000)&(abs(para[i]-temp[i])/(abs(temp[i])) > epsilon) ) {
    temp[i] <- para[i]
    para[i] <- para[i] - log_cos_D1(para[i])/log_cos_D2(para[i])
    if(abs(para[i]) == Inf) {break}
    iter <- iter + 1
    #print(para[i])
  }
}
print(para)
```

```
##   [1] -3.1421470 -3.1099375 -3.0769175 -3.0438883 -2.9862293 -2.9808953
##   [7] -2.9503023 -2.9196235 -2.8886503 -2.8573610 -2.8258534 -2.7942822
##  [13] -2.7627067 -2.7311387 -2.6996089 -2.6679818 -2.6363612 -2.6048332
##  [19] -2.5732648 -2.5417143 -2.5101200 -2.4784189 -2.4469208 -2.4153942
##  [25] -2.3838213 -2.3522535 -2.3207197 -2.2890875 -2.2575248 -2.2259526
##  [31] -2.1944008 -2.1630228 -2.1320151 -2.1015594 -2.0984893 -2.0979572
##  [37] -2.0983081 -2.0991544 -2.0993606 -2.0982989 -2.0995181 -2.0987066
##  [43] -2.0994312 -2.0991218 -2.0985298 -2.0983966 -1.6903913 -1.6574795
##  [49] -1.6254471 -1.5938975 -1.5625352 -1.5311748 -1.4997261 -1.4681814
##  [55] -1.4366077 -1.4050512 -1.3736357 -1.3425156 -1.3186971 -1.3180609
```

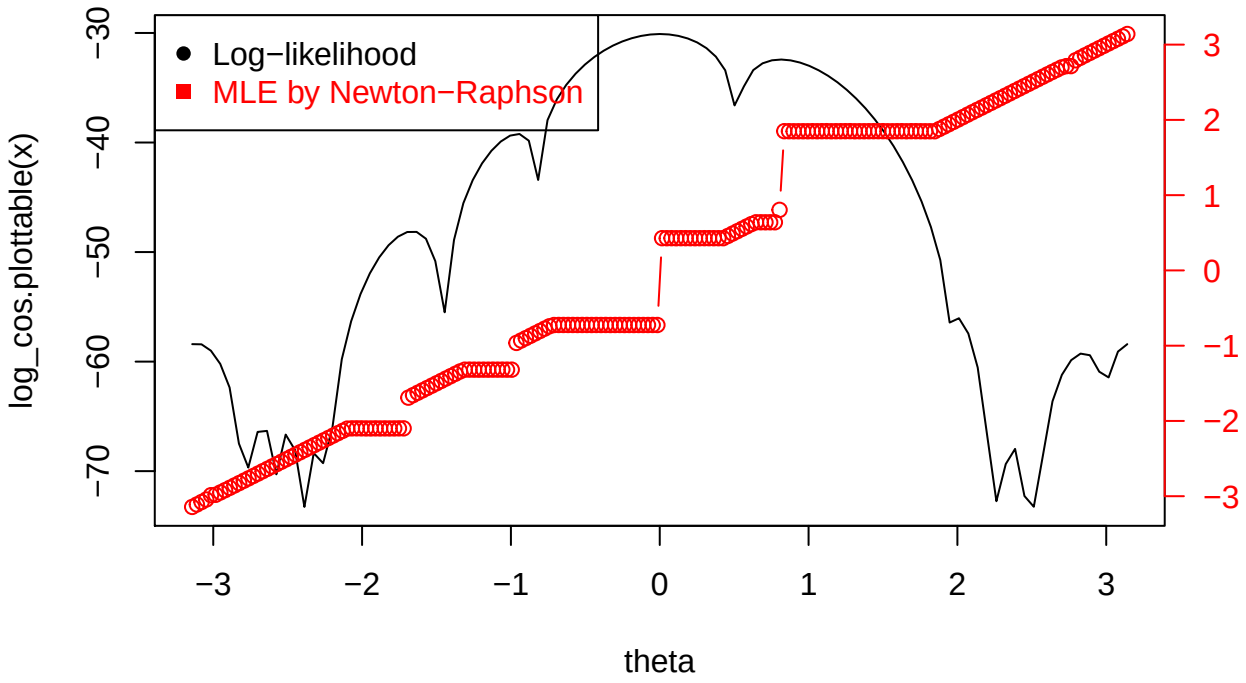
```
##  [61] -1.3182941 -1.3185565 -1.3181618 -1.3186324 -1.3180132 -1.3181516
##  [67] -1.3179187 -1.3183730 -1.3185563 -0.9631142 -0.9312931 -0.8997462
##  [73] -0.8682469 -0.8367044 -0.8051350 -0.7736355 -0.7423622 -0.7254113
##  [79] -0.7247663 -0.7247849 -0.7248769 -0.7249449 -0.7250450 -0.7248600
##  [85] -0.7249321 -0.7251640 -0.7247881 -0.7251865 -0.7249138 -0.7253687
##  [91] -0.7251252 -0.7248758 -0.7253206 -0.7250327 -0.7254081 -0.7250131
##  [97] -0.7252162 -0.7252278 -0.7248338 -0.7252505  0.4283557  0.4280697
## [103]  0.4283589  0.4279754  0.4283668  0.4283586  0.4280268  0.4283044
## [109]  0.4280597  0.4283605  0.4283655  0.4281215  0.4281008  0.4279924
## [115]  0.4579482  0.4894087  0.5209676  0.5525251  0.5839990  0.6153368
## [121]  0.6397172  0.6396775  0.6394579  0.6395375  0.6395949  0.8046583
## [127]  1.8501991  1.8497966  1.8490294  1.8489455  1.8493700  1.8490720
## [133]  1.8501415  1.8492247  1.8498832  1.8503813  1.8491017  1.8495115
## [139]  1.8498470  1.8501305  1.8503907  1.8489066  1.8491647  1.8494102
## [145]  1.8496399  1.8498559  1.8500669  1.8502871  1.8487940  1.8491130
## [151]  1.8495217  1.8500669  1.8490903  1.8502084  1.8501620  1.8493903
## [157]  1.8488009  1.8488807  1.8489031  1.8792375  1.9103261  1.9417924
## [163]  1.9733613  2.0049335  2.0366779  2.0689210  2.1009061  2.1319497
## [169]  2.1630079  2.1943941  2.2259525  2.2575309  2.2890998  2.3206470
## [175]  2.3521779  2.3838648  2.4154385  2.4469704  2.4785414  2.5101204
## [181]  2.5416905  2.5732348  2.6046310  2.6357777  2.6666448  2.6973277
## [187]  2.7140480  2.7145678  2.7922908  2.8252431  2.8575600  2.8892428
## [193]  2.9206980  2.9521743  2.9837238  3.0152893  3.0467834  3.0781594
## [199]  3.1094882  3.1410383
```

```r
#par(pty="s")
curve(log_cos.plottable, from = -pi, to = pi, xlab="theta")
## Allow a second plot on the same graph
par(new=TRUE)

## Plot the second plot and put axis scale on right
plot(seq(-pi, pi, length.out = 200), para, pch=1,  xlab="", ylab="",
    axes=FALSE, type="b", col="red")

## a little farther out (line=4) to make room for labels
mtext("MLE",side=3,col="red",line=4)
axis(4, ylim=c(-4,4), col="red",col.axis="red",las=1)


## Add Legend
legend("topleft",legend=c("Log-likelihood","MLE by Newton-Raphson"),
  text.col=c("black","red"),pch=c(16,15),col=c("black","red"))
```




```
#par(pty="m")
```

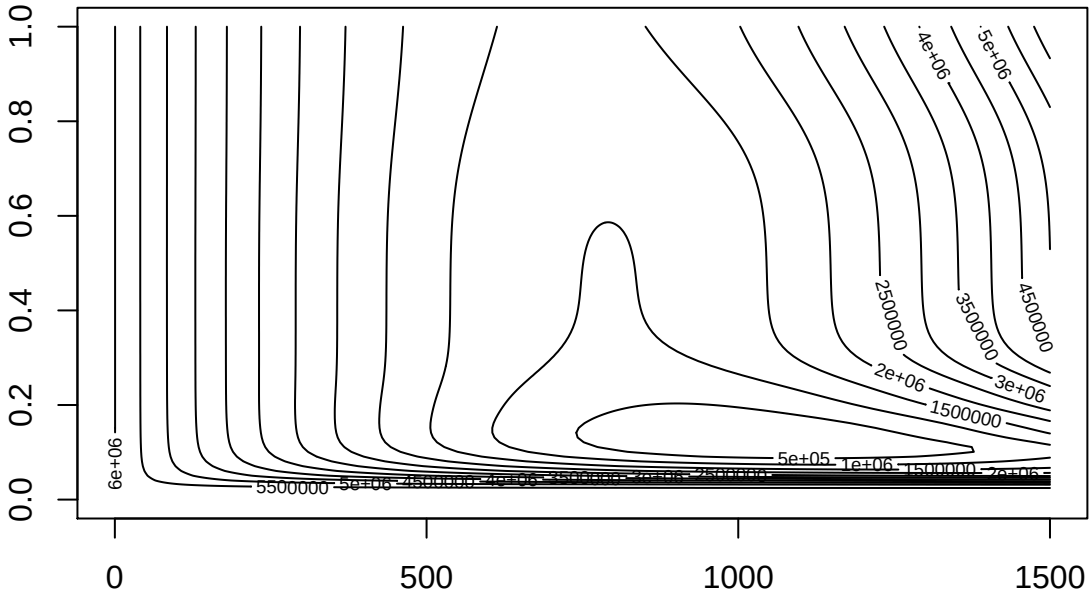
3.3.3 Modeling beetle data

```
beet <- 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))

growth <- function(x) {
  k <- x[1]; r <- x[2]
  k*beet$beetles[1]/(beet$beetles[1] + (k-beet$beetles[1])*exp(-r*beet$days))
}

growth.contour <- function(k,r) {
  k*beet$beetles[1]/(beet$beetles[1] + (k-beet$beetles[1])*exp(-r*beet$days))
}
growth.error <- function(k, r) {
  sum( (beet$beetles - growth.contour(k, r))^2 )
}
```

Plot the contour plot to visually check:



Gauss-Newton method

```
library(rootSolve)

#gradient(growth, c(1000, 0.11))

para <- c(500, 0.2);  temp <- c(0, 0); iter <- 0; epsilon <- 1e-5

while( iter <= 10000  &  (sum(para-temp)^2)^0.5 > epsilon ) {
  temp <- para
  grad <- gradient(growth, para)
  para <- para + solve( (t(grad)%*%grad)+diag(0.0001, 2) )%*%t(grad)%*%(beet$beetles-growth(para))
  if(any(para == Inf)) {break}
  iter <- iter + 1
  # print(para)
}

print(para)
```

```
##              [,1]
## [1,] 1049.4072416
## [2,]    0.1182684
```

```
#/(sum(temp^2))^0.5
```

Above is my own Gauss-Newton method, which use the formula to calculate the optimal parameters, but this depends on the inital values of parameters, so it's not very effect compared with default "nls" function in R

```
nls.R <- nls(beetles ~ growth.contour(k,r), data = beet, start = list(k = 100, r = 0.5))
summary(nls.R)
```

```
##
## Formula: beetles ~ growth.contour(k, r)
##
```

```
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## k 1.049e+03  4.717e+01   22.25 1.76e-08 ***
## r 1.183e-01  6.533e-03   18.10 8.90e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 95.8 on 8 degrees of freedom
##
## Number of iterations to convergence: 10
## Achieved convergence tolerance: 2.188e-06
```

The default method for nlw uses Gauss-Newton, we can see that the result is pretty robust compare with my own Gauss-Newton method

Finally, use self method: Steepest Descent(Gradient Descent)

```
growth.error.optim <- function(x) {
  sum( (beet$beetles - growth(x))^2 )
}

iter_num <- 2000
para <- matrix(0, nrow = iter_num, ncol=2, byrow = T)
para[1, ] <- c(100, 0.8); para[2, ] <- para[1,]-0.000001*gradient(growth.error.optim, para[1, ])
iter <- 2
epsilon <- 1e-10

while( iter <= iter_num  &  (sum(para[iter, ]-para[iter-1, ])^2)^0.5 > epsilon ) {

  grad.now <- gradient(growth.error.optim, para[iter,])
  grad.old <- gradient(growth.error.optim, para[iter-1,])
  cat("This is Hessian: ", hessian(growth.error.optim, para[iter,]), "\n")
  l <- (para[iter, ]-para[iter-1,]) %*% t(grad.now-grad.old)/sum((grad.now-grad.old)^2)
  cat("This is l: ", l, "\n")
  para[iter+1, ] <- para[iter, ] - l %*% grad.now

  if(any(para[iter, ] == Inf)) {break}
  iter <- iter + 1
  cat("This current parameter: ", para[iter, ], "\n")
}
```

```
## This is Hessian:  1041.057 0 0 0
## This is l:  -3.513244e-05
## This current parameter:  99.62471 0.9780046
## This is Hessian:  1048.571 93483.09 93483.09 9313226
## This is l:  -4.999659e-05
## This current parameter:  99.07572 1.053926
## This is Hessian:  -2735.447 -89191.4 -89191.4 -8385355
## This is l:  -0.000103967
## This current parameter:  97.93345 1.141048
## This is Hessian:  112.3749 0 0 -356.7949
## This is l:  -8.877097e-05
## This current parameter:  96.9565 1.177189
## This is Hessian:  1104.392 81597.55 81597.55 6720349
## This is l:  0.0009697241
## This current parameter:  107.6451 0.8915643
```

```
## This is Hessian:  -1506.707 0 0 9313226
## This is l:  7.382176e-05
## This current parameter:  108.4458 0.6135798
## This is Hessian:  1684.376 0 0 0
## This is l:  -4.023235e-05
## This current parameter:  108.007 1.049741
## This is Hessian:  -698.1093 0 0 -1146.873
## This is l:  -4.562408e-05
## This current parameter:  107.513 1.104669
## This is Hessian:  -704.937 78416.39 78416.39 7631254
## This is l:  -0.0001092481
## This current parameter:  106.3294 1.189304
## This is Hessian:  -721.6687 0 0 -321.91
## This is l:  -6.496408e-05
## This current parameter:  105.6243 1.214175
## This is Hessian:  -731.905 0 0 6317131
## This is l:  0.001144861
## This current parameter:  118.0646 0.8630887
## This is Hessian:  -577.7748 0 0 0
## This is l:  9.595738e-06
## This current parameter:  118.1669 0.8022475
## This is Hessian:  757.3521 0 0 0
## This is l:  -2.211701e-05
## This current parameter:  117.9307 1.003765
## This is Hessian:  -579.3173 -78675.83 -78675.83 -9245847
## This is l:  -2.997837e-05
## This current parameter:  117.6114 1.074448
## This is Hessian:  90.59011 0 0 -1271.33
## This is l:  -7.070559e-05
## This current parameter:  116.858 1.17103
## This is Hessian:  -590.7327 0 0 6790939
## This is l:  -0.0001162961
## This current parameter:  115.6177 1.24419
## This is Hessian:  -1300.99 0 0 6015990
## This is l:  6.046475e-05
## This current parameter:  116.2639 1.22366
## This is Hessian:  780.8021 0 0 -6220166
## This is l:  0.001241046
## This current parameter:  129.5119 0.7175673
## This is Hessian:  81.20285 -71910.2 -71910.2 -9313226
## This is l:  -2.226862e-05
## This current parameter:  129.2777 1.111472
## This is Hessian:  -476.4657 0 0 -1298.862
## This is l:  -2.431155e-05
## This current parameter:  129.0238 1.14657
## This is Hessian:  640.4217 0 0 -951.4265
## This is l:  -9.268781e-05
## This current parameter:  128.0554 1.247681
## This is Hessian:  81.70438 -58290.62 -58290.62 -5983029
## This is l:  -0.0001256406
## This current parameter:  126.7409 1.30788
## This is Hessian:  82.73324 0 0 -218.9264
## This is l:  0.0004938215
## This current parameter:  131.919 1.166485
```

```
## This is Hessian:  78.79763 0 0 6843640
## This is l:  0.0006908814
## This current parameter:  139.1006 0.4685445
## This is Hessian:  75.08687 0 0 9313226
## This is l:  -0.0003928937
## This current parameter:  134.997 1.725997
## This is Hessian:  1098.639 0 0 -7.377888
## This is l:  -0.0004377841
## This current parameter:  130.472 1.731572
## This is Hessian:  79.84504 -41223.26 -41223.26 -3106133
## This is l:  0.05552672
## This current parameter:  708.9239 1.131287
## This is Hessian:  14.12532 2903.14 2903.14 -242979.9
## This is l:  7.623705e-05
## This current parameter:  708.9361 -19.82475
## This is Hessian:  0 0 0 0
## This is l:  7.623705e-05
## This current parameter:  708.9361 -19.82475
```

```
print(para[iter, ])
```

```
## [1] 708.93614 -19.82475
```

The final gradient descent estimate for K, r are . My own Gradient Descent worked not very well, due to nearly singularity of Hessian matrix.

Multivariate method based on log-normal model MLE: use BFGS

```
m_log.mle.function <- function(x) {
  -sum( dnorm(log(beet$beetles), mean = log(growth(x[1:2])), sd = x[3], log = TRUE) )
}
optim(c(1000, 0.1, 1), m_log.mle.function, lower=c(0,0,0), upper=c(Inf, 1, Inf), method = "L-BFGS-B")$pa
```

```
## [1] 672.0908017   0.4004608   0.5858714
```

Above result is MLE for (K, r, σ^2). But actually this MLE method depends heavily on the initial value, I tried several, but could not get to the close result with Gauss-Newton one.