# Statistical Computing Homework 3, Chapter 3

*Ziqi Yang*

*20 September, 2018*

## Contents



## MLE of Cauchy Distribution

### (a) log-likelihood

**log-likelihood and their derivatives are obvious, now consider compute the Fisher information:

$$I_n(\theta) = nE[l_\theta^{'}]^2$$

where $l_\theta^{'}$ is only based on one sample point. So we have:

$$\begin{aligned} I_n(\theta) = nE[l_\theta^{'}]^2 = nE\big[4\frac{(\theta - X)^2}{(1+(\theta-X)^2)^2}\big] &= 4n\int_{-\infty}^{\infty} \frac{(\theta-X)^2}{(1+(\theta-X)^2)^2}\frac{1}{\pi(1+(\theta-X)^2)}dX \\ &= \frac{4n}{\pi}\int_{-\infty}^{\infty}\frac{(\theta-X)^2}{(1+(\theta-X)^2)^3}dX \end{aligned}$$

So let $u = \theta - X$, we have:

$$I_n(\theta) = \frac{4n}{\pi}\int_{-\infty}^{\infty}\frac{u^2}{(1+u^2)^3}du = \frac{8n}{\pi}\int_0^{\infty}\frac{u^2}{(1+u^2)^3}du$$

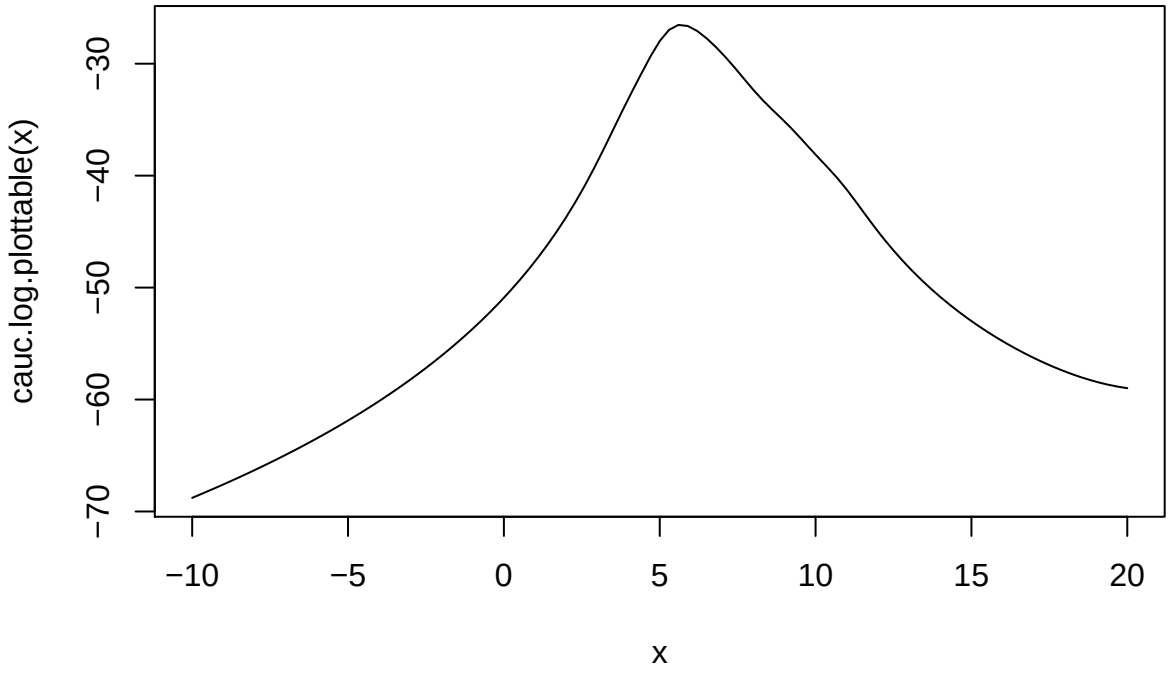
Now let $u = \tan(\theta)$, so $\theta$ is from 0 to $\pi/2$, and also we have $du = (1+\tan^2(\theta))d\theta$, so it's equal to

$$\begin{aligned} &\frac{8n}{\pi}\int_0^{\pi/2}\frac{\tan^2\theta}{(1+\tan^2\theta)^3}(1+\tan^2\theta)d\theta \\ &= \frac{8n}{\pi}\int_0^{\pi/2}\frac{\tan^2\theta}{(1+\tan^2\theta)^2}d\theta \\ &= \frac{8n}{\pi}\int_0^{\pi/2}\sin^2\theta\cos^2\theta d\theta \\ &= \frac{2n}{\pi}\int_0^{\pi/2}(2\sin\theta\cos\theta)^2 d\theta \\ &= \frac{2n}{\pi}\int_0^{\pi/2}(\sin 2\theta)^2 d\theta \\ &= \frac{2n}{\pi}\int_0^{\pi/2}\frac{1-\cos(4\theta)}{2}d\theta = \frac{n}{2} \end{aligned}$$

(b)

```
set.seed(20180909)
n <- 10; theta <- 5
cauc <- rcauchy(n, location = theta, scale = 1)
cauc.log <- function(para) {
  -n * log(pi) - sum( log(1 + (para-cauc)^2) )
}

# Use sapply here to solve the unplottable issue:
cauc.log.plottable <- function(x) {return(sapply(x, cauc.log))}
curve(cauc.log.plottable(x), from = -10, to = 20)
```




(c) MLE using Newton-Raphson

```
cauc.log.D1 <- function(para) {
  -2 * sum( (para - cauc)/(1+(para-cauc)^2) )
}

# cauc.log.D1.plottable <- function(x) {return(sapply(x, cauc.log.D1))}
# curve(cauc.log.D1.plottable(x), from = -20, to = 0)

cauc.log.D2 <- function(para) {
  -2 * sum( (1-(para - cauc)^2)/(1+(para-cauc)^2)^2 )
}
```

```
para <- seq(-10,20,by=0.5);  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] - cauc.log.D1(para[i])/cauc.log.D2(para[i])
    if(abs(para[i]) == Inf) {break}
    iter <- iter + 1
    #print(para[i])
  }
}
print(para)
```

```
##  [1]       Inf       Inf       Inf       Inf       Inf       Inf       Inf
##  [8]       Inf       Inf       Inf       Inf       Inf       Inf       Inf
## [15]       Inf       Inf       Inf       Inf       Inf       Inf       Inf
## [22]       Inf       Inf       Inf       Inf       Inf       Inf      -Inf
## [29] 20.563657 21.082295  5.685418  5.685422  5.685422  5.685421      -Inf
## [36]       Inf 19.377436 21.082295  5.685422       Inf 21.082295  5.685411
## [43]      -Inf      -Inf 20.563657 20.563657 20.563657 21.082295 21.082295
## [50] 21.082295 19.377436 20.563657 20.563657 19.377436 19.377436 19.377436
## [57]       Inf 20.563657 20.563657 20.563657 20.563657
```

** From the result, we can see that for some starting points, the algorithm won't converge, some other converge to different values, some converge to MLE.**

## (d) Fixed-point iteration:

**When $\alpha = 1$, we have the final results:**

```
alpha <- 1;  para <- seq(-10,20,by=0.5); temp <- rep(0, length(para))
G <- function(x) {
  alpha * cauc.log.D1(x) + x
}

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] <- G(para[i])
    if(abs(para[i]) == Inf) {break}
    iter <- iter + 1
    # print(para[i])
  }
}
print(para)
```

```
##  [1] 9.547862 6.486114 9.547862 4.087057 6.486114 9.547862 4.087057
##  [8] 4.087057 6.486114 9.547862 9.547862 4.087057 6.486114 6.486114
## [15] 4.087057 9.547862 4.087057 4.087057 9.547862 6.486114 9.547862
## [22] 9.547862 9.547862 6.486114 4.087057 4.087057 4.087057 6.486114
## [29] 6.486114 6.486114 6.486114 4.087057 9.547862 9.547862 9.547862
## [36] 9.547862 9.547862 9.547862 4.087057 4.087057 4.087057 4.087057
## [43] 4.087057 4.087057 9.547862 6.486114 6.486114 6.486114 6.486114
## [50] 9.547862 9.547862 9.547862 4.087057 4.087057 4.087057 6.486114
## [57] 6.486114 9.547862 4.087057 4.087057 9.547862
```

**When $\alpha = 0.64$, we have the final results:**

```
##  [1] 7.158643 5.357011 7.221250 7.554572 4.950386 7.155533 4.953697
##  [8] 4.953101 7.610535 7.413832 7.332594 5.519007 6.112807 6.869111
## [15] 7.422711 5.554644 5.214955 7.581518 7.235865 5.967612 5.428120
## [22] 5.087242 5.213069 5.145663 7.158199 5.161538 5.980249 6.232636
## [29] 7.683861 5.154588 7.729479 6.603066 5.117521 5.063899 5.041734
## [36] 5.544034 7.678946 6.307375 7.257672 7.629820 5.291916 5.435213
## [43] 5.205032 5.150064 5.457870 7.637868 7.707446 5.957830 4.952991
## [50] 5.162371 7.364806 7.166193 5.172493 5.211792 7.698177 6.106835
## [57] 4.975519 6.915049 5.194575 6.312018 6.118838
```

**When $\alpha = 0.25$, we have the final results:**

```
##  [1] 5.684854 5.686251 5.686477 5.684832 5.684820 5.686764 5.684840
##  [8] 5.684874 5.686058 5.684822 5.686846 5.684869 5.686162 5.686047
## [15] 5.686146 5.686122 5.686030 5.686343 5.684819 5.685986 5.684871
## [22] 5.686163 5.686376 5.686461 5.686908 5.686530 5.684497 5.686037
## [29] 5.684824 5.684880 5.686057 5.686652 5.686442 5.684825 5.686442
## [36] 5.686373 5.684837 5.686375 5.686531 5.684822 5.686510 5.684365
## [43] 5.684879 5.685143 5.684824 5.686178 5.686026 5.684844 5.684832
## [50] 5.684870 5.684859 5.684819 5.686646 5.684043 5.684820 5.686169
## [57] 5.686250 5.684265 5.684819 5.686840 5.684390
```

## (e) Fisher Scoring:

```
para <- seq(-10,20,by=0.5);  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] + cauc.log.D1(para[i])/5
    if(abs(para[i]) == Inf) {break}
    iter <- iter + 1
    #print(para[i])
  }
}
print(para)
```

```
##  [1] 5.685493 5.685607 5.685631 5.685621 5.685644 5.685037 5.685643
##  [8] 5.685319 5.685453 5.685592 5.685159 5.685534 5.685227 5.685614
## [15] 5.685165 5.685465 5.685162 5.685232 5.685303 5.685253 5.685160
## [22] 5.685370 5.685494 5.685652 5.685159 5.685233 5.685295 5.685316
## [29] 5.685174 5.685564 5.685198 5.685621 5.685640 5.685649 5.685594
## [36] 5.685482 5.685368 5.685509 5.685600 5.685649 5.685601 5.685520
## [43] 5.685429 5.685648 5.685537 5.685043 5.685387 5.685461 5.685360
## [50] 5.685466 5.685575 5.685622 5.685519 5.685476 5.685642 5.685640
## [57] 5.685352 5.685643 5.685651 5.685645 5.685397
```

**It's convergent time is pretty fast and stable**

## (f) Comments

- Newton-Raphson is not very stable here, it severely depends on the starting points. It's also not very fast

- Fixed point method is stable and fast, if we choose the correct $\alpha$
- Fisher scoring is stable and fast