Optimization in Differnet Methods

Yuance He

09/20/2018

1.

The density function is:

$$f(x;\theta) = \frac{1}{\pi[1+(x-\theta)^2]}$$

Likelihood function of θ is:

$$\prod_{i=1}^{n} f(x_i;\theta)$$

Log-likelihood function of θ is:

$$l(\theta) = \sum_{i=1}^{n} \ln(\frac{1}{\pi[1+(x_i-\theta)^2]})$$

$$l(\theta) = -n\ln\pi - \sum_{i=1}^{n} \ln[1+(x_i-\theta)^2]$$

First derivative of log-likelihood function of θ is:

$$l'(\theta) = -2\sum_{i=1}^{n} \frac{\theta - x_i}{1+(\theta-x_i)^2}$$

Second derivative of log-likelihood function of θ is:

$$l''(\theta) = -2\sum_{i=1}^{n} \frac{1+(\theta-x_i)^2 - 2(\theta-x_i)^2}{[1+(\theta-x_i)^2]^2}$$

$$l''(\theta) = -2\sum_{i=1}^{n} \frac{1-(\theta-X_i)^2}{[1+(\theta-x_i)^2]^2}$$

Fisher information of θ is: $I(\theta) = -E[l''(\theta)]$

$$I(\theta) = 2nE[\frac{1-(\theta-x_i)^2}{[1+(\theta-x_i)^2]^2}]$$

$$I(\theta) = \frac{4n}{\pi} \int_{-\infty}^{\infty} \frac{(x-\theta)^2}{[1+(x-\theta)^2]^3} dx$$

$$I(\theta) = \frac{4n}{\pi} \int_{-\infty}^{\infty} \frac{x^2}{[1+x^2]^3} dx$$

Let $x = tan\alpha, dx = \frac{1}{cos^2\alpha} d\alpha$,

$$I(\theta) = \frac{4n}{\pi} \int_{-\pi/2}^{\pi/2} sin^2 2\alpha d\alpha$$

Finally,

$$I(\theta) = \frac{n}{2}$$

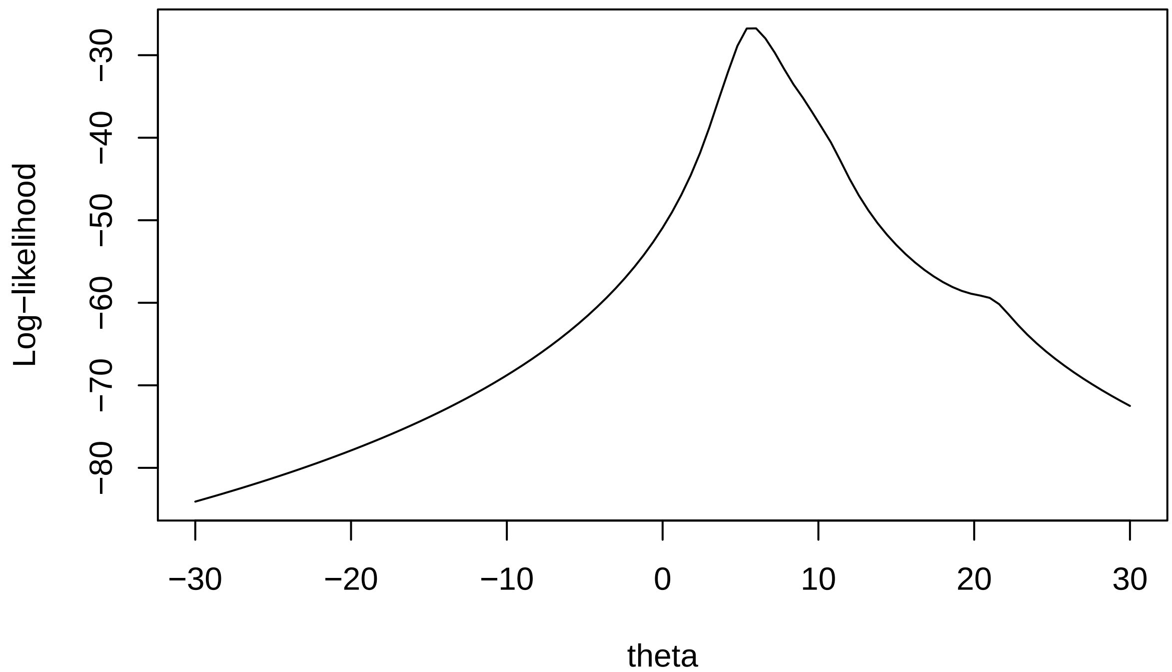
2.

```r
set.seed(20180909)
C=rcauchy(10,5)
C
```

```
##  [1]  5.327895  5.255225  5.311910  5.666190  7.343252  6.476061  9.140347
##  [8]  3.849958 10.825089 21.291087
```

```r
lg <- function(theta){
  l <- sum(dcauchy(C, location = theta, log= TRUE))
  return(l)
}

lf <- Vectorize(lg)
curve(lf, from = -30, to = 30, xname = "theta", ylab = "Log-likelihood")
```



3.

```r
set.seed(20180909)
c <- rcauchy(10,5)

lg1 <- function(theta){
  first=-2*sum((theta-c)/(1+(theta-c)^2))
  return(first)
}
lg2 <- function(theta){
  second=-2*sum((1-(theta-c)^2)/(1+(theta-c)^2)^2)
  return(second)
}

start=seq(-10, 20, by = 0.5)
Newton <- function(start, max, tol = 1e-5){
  sp = start
   for(i in 1:max)
   {
      update = sp - lg1(sp)/lg2(sp)
      if(abs(update -sp) < tol) break
```

```r
      sp = update
   }
  return( c(sp, i ) )
}


result = matrix(0, 61, 2)
for(i in 1:61)
{
   result[i,] = Newton(start[i], 100)
}
colnames(result) = c('Root', '# of iteration')
rownames(result) = c(seq(-10, 20, by = 0.5))

knitr::kable(result)
```

	Root	# of iteration
-10	-2.162370e+31	100
-9.5	-2.096788e+31	100
-9	-2.031124e+31	100
-8.5	-1.965374e+31	100
-8	-1.899532e+31	100
-7.5	-1.833592e+31	100
-7	-1.767550e+31	100
-6.5	-1.701398e+31	100
-6	-1.635130e+31	100
-5.5	-1.568739e+31	100
-5	-1.502218e+31	100
-4.5	-1.435559e+31	100
-4	-1.368755e+31	100
-3.5	-1.301799e+31	100
-3	-1.234687e+31	100
-2.5	-1.167416e+31	100
-2	-1.099990e+31	100
-1.5	-1.032425e+31	100
-1	-9.647538e+30	100
-0.5	-8.970499e+30	100
0	-8.294609e+30	100
0.5	-7.622912e+30	100
1	-6.961979e+30	100
1.5	-6.327195e+30	100
2	-5.759618e+30	100
2.5	-5.396788e+30	100
3	-5.998749e+30	100
3.5	7.514786e+31	100
4	2.108229e+01	100
4.5	1.937745e+01	100

	Root	# of iteration
5	5.685418e+00	5
5.5	5.685422e+00	4
6	5.685422e+00	5
6.5	5.685421e+00	7
7	1.607987e+31	100
7.5	-4.779444e+30	100
8	2.056366e+01	100
8.5	1.937745e+01	100
9	5.685422e+00	7
9.5	-4.379744e+30	100
10	1.937744e+01	100
10.5	5.685422e+00	6
11	3.719780e+30	100
11.5	2.214539e+31	100
12	2.108230e+01	100
12.5	2.108229e+01	100
13	2.108230e+01	100
13.5	1.937744e+01	100
14	1.937744e+01	100
14.5	1.937743e+01	100
15	2.056366e+01	100
15.5	2.108230e+01	100
16	2.108229e+01	100
16.5	2.056366e+01	100
17	2.056366e+01	100
17.5	2.056366e+01	100
18	-1.412740e+30	100
18.5	2.108229e+01	100
19	2.108230e+01	100
19.5	2.108230e+01	100
20	2.108230e+01	100

From table above, we can clearly see that choosing appropriate starting point can easily reduce the iteration times. And for complex functions we need to increase our space of starting point, in case we could have some missing root.

4.

```
set.seed(20180909)
c <- rcauchy(10,5)

lg1 <- function(theta){
  first=-2*sum((theta-c)/(1+(theta-c)^2))
```

```
  return(first)
}

start=seq(-10, 20, by = 0.5)
Fixed <- function(start,alpha, max, tol = 1e-5){
  sp = start
   for(i in 1:max)
   {
      update = sp + (alpha*lg1(sp))
      if(abs(update -sp) < tol) break
      sp = update
   }
  return( c(sp, i ) )
}

alpha1 <- matrix(0,61,2)
alpha0.64 <- matrix(0,61,2)
alpha0.25 <- matrix(0,61,2)

for(i in 1:61)
{
   alpha1[i,] = Fixed(start[i], 1, 100)
   alpha0.64[i,]=Fixed(start[i], 0.64, 100)
   alpha0.25[i,]=Fixed(start[i], 0.25, 100)
}
colnames(alpha1) = c('Root', '# of iteration')
rownames(alpha1) = c(seq(-10, 20, by = 0.5))
colnames(alpha0.64) = c('Root', '# of iteration')
rownames(alpha0.64) = c(seq(-10, 20, by = 0.5))
colnames(alpha0.25) = c('Root', '# of iteration')
rownames(alpha0.25) = c(seq(-10, 20, by = 0.5))
knitr::kable(alpha1)
```

	Root	# of iteration
-10	4.087057	100
-9.5	9.547862	100
-9	4.087057	100
-8.5	6.486114	100
-8	9.547862	100
-7.5	4.087057	100
-7	6.486114	100
-6.5	6.486114	100
-6	9.547862	100
-5.5	4.087057	100
-5	4.087057	100
-4.5	6.486114	100
-4	9.547862	100

	Root	# of iteration
-3.5	9.547862	100
-3	6.486114	100
-2.5	4.087057	100
-2	6.486114	100
-1.5	6.486114	100
-1	4.087057	100
-0.5	9.547862	100
0	4.087057	100
0.5	4.087057	100
1	4.087057	100
1.5	9.547862	100
2	6.486114	100
2.5	6.486114	100
3	6.486114	100
3.5	9.547862	100
4	9.547862	100
4.5	9.547862	100
5	9.547862	100
5.5	6.486114	100
6	4.087057	100
6.5	4.087057	100
7	4.087057	100
7.5	4.087057	100
8	4.087057	100
8.5	4.087057	100
9	6.486114	100
9.5	6.486114	100
10	6.486114	100
10.5	6.486114	100
11	6.486114	100
11.5	6.486114	100
12	4.087057	100
12.5	9.547862	100
13	9.547862	100
13.5	9.547862	100
14	9.547862	100
14.5	4.087057	100
15	4.087057	100
15.5	4.087057	100
16	6.486114	100
16.5	6.486114	100
17	6.486114	100
17.5	9.547862	100
18	9.547862	100
18.5	4.087057	100
19	6.486114	100

	Root	# of iteration
19.5	6.486114	100
20	4.087057	100

```
knitr::kable(alpha0.64)
```

	Root	# of iteration
-10	5.480685	100
-9.5	7.689808	100
-9	5.138555	100
-8.5	5.088589	100
-8	7.184783	100
-7.5	5.216443	100
-7	5.959799	100
-6.5	6.172220	100
-6	5.142718	100
-5.5	5.121909	100
-5	5.175734	100
-4.5	7.624016	100
-4	5.301880	100
-3.5	5.566243	100
-3	4.973132	100
-2.5	7.702228	100
-2	7.641402	100
-1.5	4.974966	100
-1	5.162085	100
-0.5	5.236354	100
0	7.612730	100
0.5	7.158276	100
1	7.208280	100
1.5	6.739924	100
2	5.270786	100
2.5	7.006485	100
3	5.256228	100
3.5	5.223783	100
4	4.953024	100
4.5	7.064915	100
5	5.062436	100
5.5	5.470996	100
6	6.529974	100
6.5	6.609016	100
7	6.429923	100
7.5	7.720920	100
8	5.057480	100
8.5	5.566835	100
9	5.119058	100

	Root	# of iteration
9.5	5.162731	100
10	7.227417	100
10.5	7.348496	100
11	7.341966	100
11.5	6.168473	100
12	7.223644	100
12.5	5.150102	100
13	5.183969	100
13.5	5.198920	100
14	6.112803	100
14.5	7.155070	100
15	5.185394	100
15.5	5.283418	100
16	7.159304	100
16.5	7.479077	100
17	5.092643	100
17.5	5.538366	100
18	5.967631	100
18.5	5.300515	100
19	7.725016	100
19.5	5.226117	100
20	5.524462	100

```
knitr::kable(alpha0.25)
```

	Root	# of iteration
-10	5.685425	41
-9.5	5.685417	39
-9	5.685415	37
-8.5	5.685425	36
-8	5.685425	35
-7.5	5.685424	33
-7	5.685425	32
-6.5	5.685425	30
-6	5.685418	28
-5.5	5.685425	28
-5	5.685425	24
-4.5	5.685425	25
-4	5.685417	23
-3.5	5.685418	22
-3	5.685417	21
-2.5	5.685417	20
-2	5.685418	19
-1.5	5.685416	18
-1	5.685425	18

	Root	# of iteration
-0.5	5.685418	17
0	5.685425	17
0.5	5.685417	15
1	5.685416	13
1.5	5.685416	14
2	5.685425	14
2.5	5.685415	13
3	5.685427	11
3.5	5.685418	12
4	5.685425	13
4.5	5.685425	12
5	5.685418	11
5.5	5.685414	11
6	5.685416	10
6.5	5.685425	12
7	5.685416	12
7.5	5.685416	12
8	5.685425	14
8.5	5.685416	14
9	5.685415	12
9.5	5.685425	16
10	5.685415	16
10.5	5.685428	15
11	5.685425	18
11.5	5.685417	14
12	5.685425	19
12.5	5.685417	19
13	5.685418	19
13.5	5.685425	21
14	5.685425	22
14.5	5.685425	23
15	5.685425	24
15.5	5.685425	25
16	5.685414	25
16.5	5.685419	27
17	5.685425	29
17.5	5.685417	30
18	5.685417	30
18.5	5.685429	33
19	5.685425	37
19.5	5.685425	40
20	5.685428	43

Basically there is not any convergence among 61 starting point when we take alpha equal to 1 or 0.64 in 100 times iterations. However, when alpha comes to 0.25, which means that we increase the speed of convergence, we can easily get convergency root 5.685 under no more that 45 times of iteration.

5.

From section 1, we have Fisher information of θ is: $I(\theta) = \frac{n}{2}$. Then we take a subsitution of $l''(\theta)$ by $\frac{n}{2}$ in Newton's method.

```
set.seed(20180909)
c <- rcauchy(10,5)

lg1 <- function(theta){
  first=-2*sum((theta-c)/(1+(theta-c)^2))
  return(first)
}

start=seq(-10, 20, by = 0.5)
Newton <- function(start, max, tol = 1e-5){
  sp = start
   for(i in 1:max)
   {
      update = sp + lg1(sp)/5
      if(abs(update -sp) < tol) break
      sp = update
   }
  return( c(sp, i ) )
}


result = matrix(0, 61, 2)
for(i in 1:61)
{
   result[i,] = Newton(start[i], 100)
}
colnames(result) = c('Root', '# of iteration')
rownames(result) = c(seq(-10, 20, by = 0.5))

knitr::kable(result)
```

	Root	# of iteration
-10	5.685416	42
-9.5	5.685423	40
-9	5.685423	38
-8.5	5.685423	36
-8	5.685423	34
-7.5	5.685419	32
-7	5.685423	31

	Root	# of iteration
-6.5	5.685431	28
-6	5.685419	26
-5.5	5.685423	26
-5	5.685420	24
-4.5	5.685422	23
-4	5.685420	21
-3.5	5.685423	20
-3	5.685420	18
-2.5	5.685418	16
-2	5.685420	16
-1.5	5.685420	15
-1	5.685421	14
-0.5	5.685420	13
0	5.685420	12
0.5	5.685426	10
1	5.685415	9
1.5	5.685423	10
2	5.685420	9
2.5	5.685420	8
3	5.685421	8
3.5	5.685421	7
4	5.685420	7
4.5	5.685423	7
5	5.685420	6
5.5	5.685423	6
6	5.685423	6
6.5	5.685423	7
7	5.685423	8
7.5	5.685416	8
8	5.685426	8
8.5	5.685414	9
9	5.685423	11
9.5	5.685423	12
10	5.685423	13
10.5	5.685413	13
11	5.685429	12
11.5	5.685423	15
12	5.685422	16
12.5	5.685419	16
13	5.685425	16
13.5	5.685418	17
14	5.685427	18
14.5	5.685418	20
15	5.685423	22
15.5	5.685423	23
16	5.685413	24

	Root	# of iteration
16.5	5.685417	25
17	5.685423	28
17.5	5.685423	30
18	5.685428	31
18.5	5.685423	35
19	5.685423	38
19.5	5.685423	42
20	5.685424	46

By using Fisher information, there is a significant reduction of iteration times than Newton' method. It does not only improve the efficency, also the accuracy.

6.

In my opinion, Newton's method is the most general basic way to compute roots of a function. It is very kind for me to understand the theory behind the metheod. However, in contrast to Fixed point and Fisher information, the Newton's method only has a "OK" performance. Fixed point and Fisher information methods have higher speed and less depend on starting point, which means we can use them to compute even more complex functions.