# Cauchy Project Stat5630 HW3

*Yiyi Xu*

*9/18/2018*

# Part 1 Math Equation

$$\begin{aligned} L(\theta; x) &= \prod_{i=1}^{n} f(x_i; \theta) \quad \theta \in \Omega \\ &= \prod_{i=1}^{n} \frac{1}{\pi[1 + (x - \theta)^2]} \\ &= (\frac{1}{\pi})^n \cdot \prod_{i=1}^{n} [1 + (x - \theta)^2]^{-1} \end{aligned}$$

$$\begin{aligned} l(\theta) &= \ln L(\theta; X) \\ &= \ln((\frac{1}{\pi})^n \prod_{i=1}^{n} \frac{1}{[1 + (x_i - \theta)^2]}) \\ &= \ln((\frac{1}{\pi})^n) + \ln(\prod_{i=1}^{n} \frac{1}{[1 + (x_i - \theta)^2]}) \\ &= -n \ln(\pi) - \sum_{i=1}^{n} \ln([1 + (x_i - \theta)^2]) \end{aligned}$$

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

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

$$\begin{aligned} I(\theta) &= -\int_{-\infty}^{\infty} -2 \sum_{i=1}^{n} \frac{1 - (\theta - x_i)^2)}{(1 + (\theta - x_i)^2))^2} \frac{1}{\pi[1 + (x - \theta)^2]} \, \mathrm{d}x \\ &= \frac{2n}{\pi} \int_{-\infty}^{\infty} \frac{1 - (\theta - x)^2)}{(1 + (\theta - x)^2))^3} \, \mathrm{d}x \\ &= \frac{2n}{\pi} \int_{-\infty}^{\infty} \frac{1 - x^2)}{(1 + x^2))^3} \, \mathrm{d}x \end{aligned}$$

$$
\begin{aligned}
&= \frac{2n}{\pi}\int_{-\infty}^{\infty} -\frac{1+x^2-2)}{(1+x^2))^3}\,\mathrm{d}x \\
&= \frac{2n}{\pi}\int_{-\infty}^{\infty} -\frac{1)}{(1+x^2))^2} + 2\frac{1)}{(1+x^2))^3}\,\mathrm{d}x \\
&= \frac{2n}{\pi}[\frac{arctan(x)}{4} + \frac{x^3+3x}{4x^4+8x^2+4}]_{-\infty}^{\infty} \\
&= \frac{2n}{\pi}\frac{\pi}{4} \\
&= \frac{n}{2}
\end{aligned}
$$

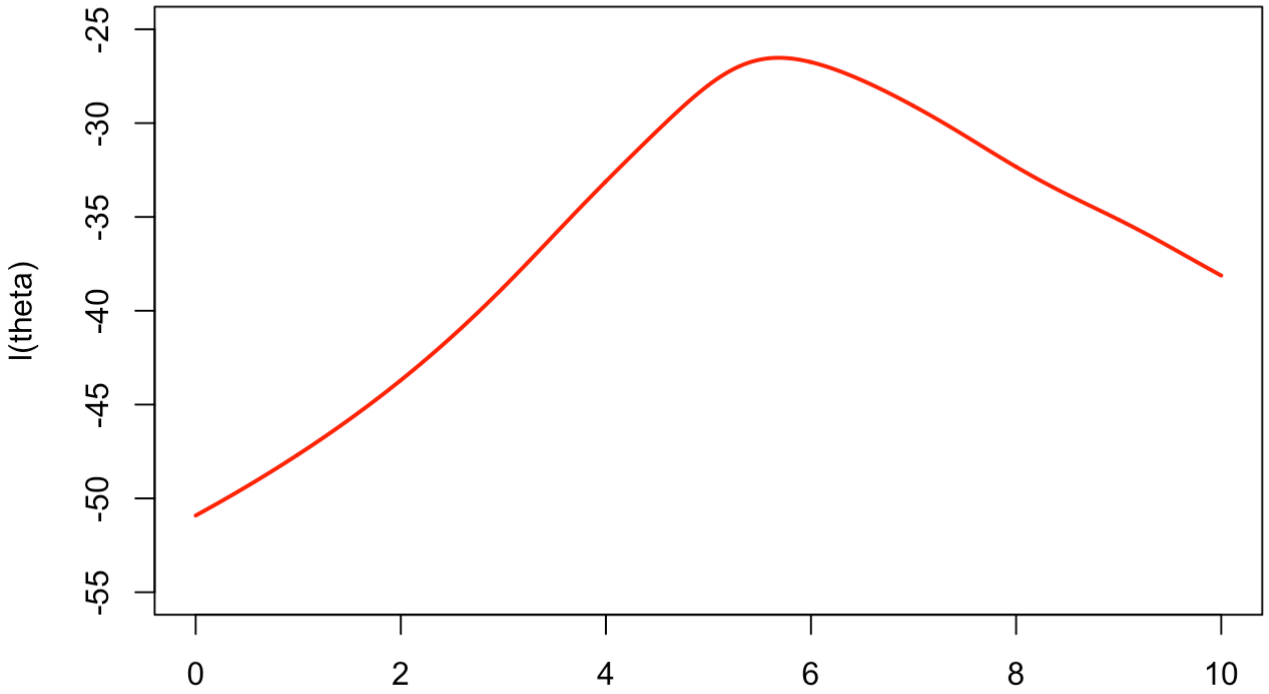
# Part 2 Figure for loglikelihood function against $\theta$

From figure, we can see the loglikellihood function against $\theta$ reach the maximize when $\theta$ near 6.

```
set.seed(20180909)
X <- rcauchy(10, location = 5, scale =1 )
n <- 10


l <- function(x, theta, n){
  - n*log(pi, base = exp(1)) - sum(log(1 + (theta - x)^2, base = exp(1)))
}


theta <- seq(0, 10, 0.1)
L <- numeric(0)
for (i in 1: length(theta)){
   L[i] <- l(X,theta[i],10)
}
 sp=spline(theta,L,n=1000)
 plot(sp,col="red",type="l",xlim=c(0,10),ylim=c(-55,-25),lwd=2,xlab="theta",ylab="l(thet
a)",sub="loglikehood function against theta",col.main="green",font.main=2)
```




loglikehood function against theta

# Part 3 the Newton-Raphson method

```
l1 <- function(x, theta, n){
  -2*sum((theta-x)/(1 + (theta - x)^2))
}
l2 <- function(x,theta,n){
  -2*sum((1- (theta-x)^2)/(((1 + (theta - x)^2))^2))
}
Newton.Method <- function(y,f,f1){
   y0 <- y
   for(i in 1:100){
    y1 <- y0 - f(X,y0,10)/f1(X,y0,10)
    if(abs(y1-y0)<0.0001)
      break
    y0 <- y1
   }
   return(data.frame(init=y,root=y1,iter=i))
}
```

# comment

From the Rood in Newton Method Table, when initial value eqauls from 20 to -10, there are only 6 numbers can reach the approximate root in 100 iterations, which are 10.5, 9.0, 6.5,6.0, 5.5 and 5.0. the approximate root is 5.6854 which is near $\theta = 5$ or $\theta = 6$. Therefore, we can get the result that Newton-Raphson Method is not

always converge to the root. The Newton-Raphson method workring for “local” convergence, so we have to start the initial value which close to the root. If we start the initial value near the root, then the Newton-Raphson Method can quickly reach the root in less ten loops. However, if we start initial value not near the root, the Newton-Raphson is not working.

Root in Newton Method

| init | root | iter |
|---|---|---|
| 20.0 | 2.108230e+01 | 100 |
| 19.5 | 2.108230e+01 | 100 |
| 19.0 | 2.108230e+01 | 100 |
| 18.5 | 2.108229e+01 | 100 |
| 18.0 | -1.412740e+30 | 100 |
| 17.5 | 2.056366e+01 | 100 |
| 17.0 | 2.056366e+01 | 100 |
| 16.5 | 2.056366e+01 | 100 |
| 16.0 | 2.108229e+01 | 100 |
| 15.5 | 2.108230e+01 | 100 |
| 15.0 | 2.056366e+01 | 100 |
| 14.5 | 1.937743e+01 | 100 |
| 14.0 | 1.937744e+01 | 100 |
| 13.5 | 1.937744e+01 | 100 |
| 13.0 | 2.108230e+01 | 100 |
| 12.5 | 2.108229e+01 | 100 |
| 12.0 | 2.108230e+01 | 100 |
| 11.5 | 2.214539e+31 | 100 |
| 11.0 | 3.719780e+30 | 100 |
| 10.5 | 5.685422e+00 | 5 |
| 10.0 | 1.937744e+01 | 100 |
| 9.5 | -4.379744e+30 | 100 |

| init | root | iter |
|---|---|---|
| 9.0 | 5.685422e+00 | 6 |
| 8.5 | 1.937745e+01 | 100 |
| 8.0 | 2.056366e+01 | 100 |
| 7.5 | -4.779444e+30 | 100 |
| 7.0 | 1.607987e+31 | 100 |
| 6.5 | 5.685422e+00 | 7 |
| 6.0 | 5.685422e+00 | 4 |
| 5.5 | 5.685422e+00 | 4 |
| 5.0 | 5.685422e+00 | 5 |
| 4.5 | 1.937745e+01 | 100 |
| 4.0 | 2.108229e+01 | 100 |
| 3.5 | 7.514786e+31 | 100 |
| 3.0 | -5.998749e+30 | 100 |
| 2.5 | -5.396788e+30 | 100 |
| 2.0 | -5.759618e+30 | 100 |
| 1.5 | -6.327195e+30 | 100 |
| 1.0 | -6.961979e+30 | 100 |
| 0.5 | -7.622912e+30 | 100 |
| 0.0 | -8.294609e+30 | 100 |
| -0.5 | -8.970499e+30 | 100 |
| -1.0 | -9.647538e+30 | 100 |
| -1.5 | -1.032425e+31 | 100 |
| -2.0 | -1.099990e+31 | 100 |
| -2.5 | -1.167416e+31 | 100 |
| -3.0 | -1.234687e+31 | 100 |

| init | root | iter |
|---|---|---|
| -3.5 | -1.301799e+31 | 100 |
| -4.0 | -1.368755e+31 | 100 |
| -4.5 | -1.435559e+31 | 100 |
| -5.0 | -1.502218e+31 | 100 |
| -5.5 | -1.568739e+31 | 100 |
| -6.0 | -1.635130e+31 | 100 |
| -6.5 | -1.701398e+31 | 100 |
| -7.0 | -1.767550e+31 | 100 |
| -7.5 | -1.833592e+31 | 100 |
| -8.0 | -1.899532e+31 | 100 |
| -8.5 | -1.965374e+31 | 100 |
| -9.0 | -2.031124e+31 | 100 |
| -9.5 | -2.096788e+31 | 100 |
| -10.0 | -2.162370e+31 | 100 |

# Part 4 Fixed Point Iterations

```
Fixed.Point <- function(x,f1,alpha){
  x0 <- x
  for(i in 1:1000){
    x1 <- alpha*f1(X,x0,10)+ x0
    if(abs(x1-x0)<0.0001)
      break
    x0 <- x1
  }
  return(data.frame(ini=x,root=x1,iter=i))
}
```

# comment

By comparing the result of different $\alpha$ value, we can see that when $\alpha$ equals 1 and 0.64, no matter where the initial value start, the result always near the real root in 1000 iterations. However, none of them reach the real root. So the result is not accurate. But when $\alpha = 0.25$, we can see all of the initial value reach the real root in less than

50 iterations. and for initial value near real value, the iteration even less than 10. So the accuracy of Fixed Point Iteration Method depends on the $\alpha$

# when alpha = 1

Root| alpha = 1

| ini | root | iter |
|---|---|---|
| 20.0 | 4.087057 | 1000 |
| 19.5 | 6.486114 | 1000 |
| 19.0 | 6.486114 | 1000 |
| 18.5 | 4.087057 | 1000 |
| 18.0 | 9.547862 | 1000 |
| 17.5 | 9.547862 | 1000 |
| 17.0 | 6.486114 | 1000 |
| 16.5 | 6.486114 | 1000 |
| 16.0 | 6.486114 | 1000 |
| 15.5 | 4.087057 | 1000 |
| 15.0 | 4.087057 | 1000 |
| 14.5 | 4.087057 | 1000 |
| 14.0 | 9.547862 | 1000 |
| 13.5 | 9.547862 | 1000 |
| 13.0 | 9.547862 | 1000 |
| 12.5 | 9.547862 | 1000 |
| 12.0 | 4.087057 | 1000 |
| 11.5 | 6.486114 | 1000 |
| 11.0 | 6.486114 | 1000 |
| 10.5 | 6.486114 | 1000 |
| 10.0 | 6.486114 | 1000 |

| ini | root | iter |
|---|---|---|
| 9.5 | 6.486114 | 1000 |
| 9.0 | 6.486114 | 1000 |
| 8.5 | 4.087057 | 1000 |
| 8.0 | 4.087057 | 1000 |
| 7.5 | 4.087057 | 1000 |
| 7.0 | 4.087057 | 1000 |
| 6.5 | 4.087057 | 1000 |
| 6.0 | 4.087057 | 1000 |
| 5.5 | 6.486114 | 1000 |
| 5.0 | 9.547862 | 1000 |
| 4.5 | 9.547862 | 1000 |
| 4.0 | 9.547862 | 1000 |
| 3.5 | 9.547862 | 1000 |
| 3.0 | 6.486114 | 1000 |
| 2.5 | 6.486114 | 1000 |
| 2.0 | 6.486114 | 1000 |
| 1.5 | 9.547862 | 1000 |
| 1.0 | 4.087057 | 1000 |
| 0.5 | 4.087057 | 1000 |
| 0.0 | 4.087057 | 1000 |
| -0.5 | 9.547862 | 1000 |
| -1.0 | 4.087057 | 1000 |
| -1.5 | 6.486114 | 1000 |
| -2.0 | 6.486114 | 1000 |
| -2.5 | 4.087057 | 1000 |

| ini | root | iter |
|---|---|---|
| -3.0 | 6.486114 | 1000 |
| -3.5 | 9.547862 | 1000 |
| -4.0 | 9.547862 | 1000 |
| -4.5 | 6.486114 | 1000 |
| -5.0 | 4.087057 | 1000 |
| -5.5 | 4.087057 | 1000 |
| -6.0 | 9.547862 | 1000 |
| -6.5 | 6.486114 | 1000 |
| -7.0 | 6.486114 | 1000 |
| -7.5 | 4.087057 | 1000 |
| -8.0 | 9.547862 | 1000 |
| -8.5 | 6.486114 | 1000 |
| -9.0 | 4.087057 | 1000 |
| -9.5 | 9.547862 | 1000 |
| -10.0 | 4.087057 | 1000 |

# when alphs = 0.64

Root| alpha = 0.64

| ini | root | iter |
|---|---|---|
| 20.0 | 5.530510 | 1000 |
| 19.5 | 5.469104 | 1000 |
| 19.0 | 7.203068 | 1000 |
| 18.5 | 5.283310 | 1000 |
| 18.0 | 6.666568 | 1000 |
| 17.5 | 5.534426 | 1000 |

| ini | root | iter |
|---|---|---|
| 17.0 | 4.970207 | 1000 |
| 16.5 | 7.237981 | 1000 |
| 16.0 | 7.157251 | 1000 |
| 15.5 | 5.200588 | 1000 |
| 15.0 | 5.127932 | 1000 |
| 14.5 | 5.979331 | 1000 |
| 14.0 | 6.396968 | 1000 |
| 13.5 | 5.584397 | 1000 |
| 13.0 | 4.964483 | 1000 |
| 12.5 | 5.004767 | 1000 |
| 12.0 | 7.607196 | 1000 |
| 11.5 | 7.109859 | 1000 |
| 11.0 | 7.224372 | 1000 |
| 10.5 | 7.582125 | 1000 |
| 10.0 | 7.385700 | 1000 |
| 9.5 | 5.009080 | 1000 |
| 9.0 | 5.168213 | 1000 |
| 8.5 | 5.470551 | 1000 |
| 8.0 | 4.981700 | 1000 |
| 7.5 | 7.693043 | 1000 |
| 7.0 | 6.120372 | 1000 |
| 6.5 | 6.087529 | 1000 |
| 6.0 | 6.023024 | 1000 |
| 5.5 | 5.379770 | 1000 |
| 5.0 | 4.950335 | 1000 |

| ini | root | iter |
|---|---|---|
| 4.5 | 5.986422 | 1000 |
| 4.0 | 4.978809 | 1000 |
| 3.5 | 5.494015 | 1000 |
| 3.0 | 5.576697 | 1000 |
| 2.5 | 7.134195 | 1000 |
| 2.0 | 5.203346 | 1000 |
| 1.5 | 5.994786 | 1000 |
| 1.0 | 7.240537 | 1000 |
| 0.5 | 6.057336 | 1000 |
| 0.0 | 7.574015 | 1000 |
| -0.5 | 5.581029 | 1000 |
| -1.0 | 5.176078 | 1000 |
| -1.5 | 5.033793 | 1000 |
| -2.0 | 7.244301 | 1000 |
| -2.5 | 7.702770 | 1000 |
| -3.0 | 5.104768 | 1000 |
| -3.5 | 5.297792 | 1000 |
| -4.0 | 5.532476 | 1000 |
| -4.5 | 7.669443 | 1000 |
| -5.0 | 5.140361 | 1000 |
| -5.5 | 5.108395 | 1000 |
| -6.0 | 5.019178 | 1000 |
| -6.5 | 6.395664 | 1000 |
| -7.0 | 6.389073 | 1000 |
| -7.5 | 5.204264 | 1000 |

| ini | root | iter |
|---|---|---|
| -8.0 | 6.470638 | 1000 |
| -8.5 | 5.046818 | 1000 |
| -9.0 | 5.181294 | 1000 |
| -9.5 | 7.484446 | 1000 |
| -10.0 | 5.203193 | 1000 |

# when alpha = 0.25

Root| alpha =0.25

| ini | root | iter |
|---|---|---|
| 20.0 | 5.685404 | 41 |
| 19.5 | 5.685445 | 37 |
| 19.0 | 5.685412 | 35 |
| 18.5 | 5.685402 | 31 |
| 18.0 | 5.685435 | 28 |
| 17.5 | 5.685434 | 28 |
| 17.0 | 5.685412 | 27 |
| 16.5 | 5.685399 | 24 |
| 16.0 | 5.685442 | 23 |
| 15.5 | 5.685412 | 23 |
| 15.0 | 5.685448 | 21 |
| 14.5 | 5.685447 | 20 |
| 14.0 | 5.685412 | 20 |
| 13.5 | 5.685412 | 19 |
| 13.0 | 5.685432 | 17 |
| 12.5 | 5.685434 | 17 |

| ini | root | iter |
|---|---|---|
| 12.0 | 5.685412 | 17 |
| 11.5 | 5.685435 | 12 |
| 11.0 | 5.685447 | 15 |
| 10.5 | 5.685404 | 13 |
| 10.0 | 5.685440 | 14 |
| 9.5 | 5.685412 | 14 |
| 9.0 | 5.685440 | 10 |
| 8.5 | 5.685437 | 12 |
| 8.0 | 5.685412 | 12 |
| 7.5 | 5.685437 | 10 |
| 7.0 | 5.685439 | 10 |
| 6.5 | 5.685412 | 10 |
| 6.0 | 5.685439 | 8 |
| 5.5 | 5.685442 | 9 |
| 5.0 | 5.685432 | 9 |
| 4.5 | 5.685447 | 9 |
| 4.0 | 5.685412 | 11 |
| 3.5 | 5.685432 | 10 |
| 3.0 | 5.685406 | 9 |
| 2.5 | 5.685440 | 11 |
| 2.0 | 5.685446 | 11 |
| 1.5 | 5.685439 | 12 |
| 1.0 | 5.685437 | 11 |
| 0.5 | 5.685434 | 13 |
| 0.0 | 5.685447 | 14 |

| ini | root | iter |
|---|---|---|
| -0.5 | 5.685396 | 14 |
| -1.0 | 5.685412 | 16 |
| -1.5 | 5.685437 | 16 |
| -2.0 | 5.685432 | 17 |
| -2.5 | 5.685433 | 18 |
| -3.0 | 5.685434 | 19 |
| -3.5 | 5.685432 | 20 |
| -4.0 | 5.685434 | 21 |
| -4.5 | 5.685447 | 22 |
| -5.0 | 5.685445 | 21 |
| -5.5 | 5.685412 | 26 |
| -6.0 | 5.685432 | 26 |
| -6.5 | 5.685447 | 27 |
| -7.0 | 5.685412 | 30 |
| -7.5 | 5.685444 | 30 |
| -8.0 | 5.685412 | 33 |
| -8.5 | 5.685412 | 34 |
| -9.0 | 5.685439 | 35 |
| -9.5 | 5.685435 | 37 |
| -10.0 | 5.685448 | 38 |

# part 5 fisher scoring

```
In <- 5
fisher.scoring <- function(x,f1,In){
  x0 <- x
  for(i in 1:1000){
    x1 <- x0 + f1(X,x0,10)/In
    if(abs(x1-x0)<0.0001)
      break
    x0 <- x1
  }
  return(x1)
}

Newton.Method2 <- function(y,f,f1){
   y0 <- y
   for(i in 1:100){
    y1 <- y0 - f(X,y0,10)/f1(X,y0,10)
    if(abs(y1-y0)<0.0001)
      break
    y0 <- y1
   }

   return(data.frame(resultOfFisher=y,iter=i,root=y1))
}
```

# Comment

Fisher Scoring and the Newton Method is the most faster and accturate way to find the root as no matter where the initial value start, it reach the root.

Root of Fisher Scoring and Newton method

| resultOfFisher | iter | root |
|---|---|---|
| 5.685424 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685428 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685417 | 1 | 5.685422 |
| 5.685422 | 1 | 5.685422 |

| resultOfFisher | iter | root |
|---|---|---|
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685418 | 1 | 5.685422 |
| 5.685427 | 1 | 5.685422 |
| 5.685418 | 1 | 5.685422 |
| 5.685425 | 1 | 5.685422 |
| 5.685419 | 1 | 5.685422 |
| 5.685422 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685429 | 1 | 5.685422 |
| 5.685422 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685414 | 1 | 5.685422 |
| 5.685426 | 1 | 5.685422 |
| 5.685416 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685421 | 1 | 5.685422 |

| resultOfFisher | iter | root |
|---|---|---|
| 5.685421 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685415 | 1 | 5.685422 |
| 5.685426 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685421 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685418 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685422 | 1 | 5.685422 |
| 5.685420 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685419 | 1 | 5.685422 |
| 5.685421 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685419 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |
| 5.685423 | 1 | 5.685422 |

| resultOfFisher | iter | root |
|---|---|---|
| 5.685423 | 1 | 5.685422 |
| 5.685416 | 1 | 5.685422 |