Estimating the local parameter of a Cauchy distribution

Xiaokang Liu

20 September 2018

Contents

1 Introduction

Consider estimationg the location parameter of a Cauchy distribution with a known scale parameter. The density function is

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

Let $X_1, \ldots, X_n$ be a random sample of size n and $l(\theta)$ the log-likelihood function of θ based on the sample.

2 Derivation of some important quantities

2.1 Log-likelihood function

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

$$f(X_1, \ldots, X_n;\theta) = \prod_{i=1}^{n} f(X_i;\theta),$$

and

$$l(x,\theta) = \ln(f(x;\theta)) = -\ln(\pi[1+(x-\theta)^2]) = -\ln(\pi) - \ln[1+(\theta-x)^2].$$

So the log-likelihood function is

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

2.2 The first derivative of $l(\theta)$

Do derivative with respect to θ with the log-likelihood function $l(\theta)$ we can get

$$l'(\theta) = 0 - 2\sum_{i=1}^{n} \frac{\theta - X_i}{1 + (\theta - X_i)^2} = -2\sum_{i=1}^{n} \frac{\theta - X_i}{1 + (\theta - X_i)^2}.$$

2.3 The second derivative of $l(\theta)$

Do derivative with respect to θ with $l'(\theta)$, since we have

$$\begin{aligned} l''(x_i; \theta) &= -2\left\{\frac{1}{1 + (\theta - x_i)^2} - \frac{2(\theta - x_i)^2}{[1 + (\theta - x_i)^2]^2}\right\} \\ &= -2\frac{1 - (\theta - x_i)^2}{[1 + (\theta - x_i)^2]^2}. \end{aligned}$$

Then we have

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

2.4 Fisher information

By the definition of Fisher information, we have

$$I(X, \theta) = -E(l''(X, \theta))$$

thus we have

$$\begin{aligned} I(X, \theta) &= 2E\left\{\frac{1 - (\theta - x)^2}{(1 + (\theta - x)^2)^2}\right\} \\ &= \frac{2}{\pi}\int_R \frac{1 - (\theta - x)^2}{(1 + (\theta - x)^2)^3} dx \\ &= -\frac{2}{\pi}\int_R \frac{1}{(1 + x^2)^2} dx + \frac{4}{\pi}\int \frac{1}{(1 + x^2)^3} dx. \end{aligned}$$

Then we consider $I_k = \int_R \frac{1}{(1+x^2)^k} dx$. We have

$$\begin{aligned} I_k &= \int_R \frac{1 + x^2}{(1 + x^2)^{k+1}} dx \\ &= I_{k+1} + \int_R \frac{x^2}{(1 + x^2)^{k+1}} dx. \end{aligned}$$

Since we have

$$\frac{d(1+x^2)^{-k}}{dx} = \frac{-2kx}{(1+x^2)^{k+1}},$$

then we can write $\int_R \frac{x^2}{(1+x^2)^{k+1}}dx$ as $\int_R \frac{2kx}{(1+x^2)^{k+1}}\frac{x}{2k}dx$ and by using integration by parts once, we have

$$I_{k+1} = I_k(1-\frac{1}{2k}).$$

So, by using this relationship and the fact that $I_1 = \pi$, we have

$$\begin{aligned} I(X,\theta) &= -\frac{2}{\pi}I_2 + \frac{4}{\pi}I_3 \\ &= -1 + 3/2 \\ &= 1/2. \end{aligned}$$

Finally, since the sample size for the i.i.d. observations from Cauchy distribution is n, the Fisher information is $\frac{n}{2}$.

3 Implementation and Results

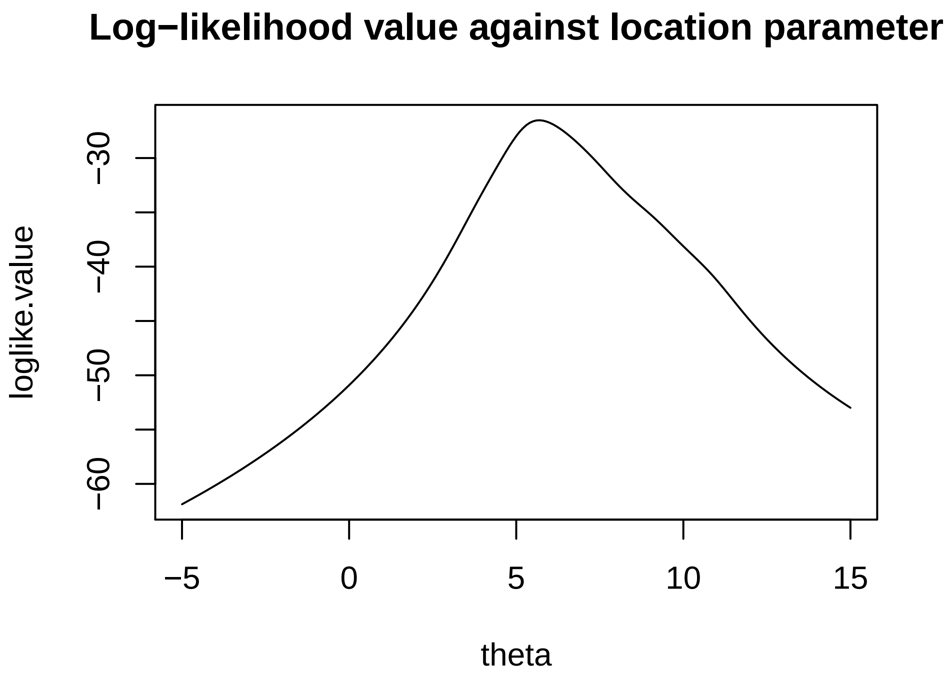
For each method, the corresponding functions are provided. In each function, the final estimate of the parameter, number of iterations needed, and an indicator of convergence are reported, with indicator= 0 indicates convergence and indicator= 1 otherwise. For comparing the efficiency of different methods, we also record the running time. The maximum iteration number is set to be 200, and the tolerance error is .Machine$double.eps^0.5.

3.1 Implement a loglikelihood function and plot against θ

```r
set.seed(20180909)
samp <- rcauchy(10,location = 5)

# log-likelihood function
l <- function(theta,x){
  val <- -length(x)*log(pi)-sum(log(1+(theta-x)^2))
  return(val)
}
# gradient
l1 <- function(theta,x){
  val <- -2*sum((theta-x)/(1+(theta-x)^2))
  return(val)
}
# Hessian
l2 <- function(theta,x){
  val <- -2*sum((1-(theta-x)^2)/(1+(theta-x)^2)^2)
  return(val)
}
```

```r
# implement a loglikelihood function and plot against \theta
the <- seq(-5,15,0.1)
loglike.value <- vector()
for (i in 1:201){
  loglike.value[i] <- l(the[i],samp)
}
plot(the, loglike.value, xlab = "theta",
     main = "Log-likelihood value against location parameter", type = "l")
```



From the plot of log-likelihood function value against location parameter value, we can see that the point at which the log-likelihood has the largest value is around 5.6.

3.2 Newton-Raphson method

```r
NR <- function(ini,x,tol,max_ite){
  err <- 100
  iter <- 0
  conver <- 0
  while ((err > tol) & (iter < max_ite)) {
    ini1 <- ini-l1(ini,x)/l2(ini,x)
    err <- abs(ini1-ini)
    ini <- ini1
    iter <- iter+1
  }
  if (iter >= max_ite) conver <- 1
  return(list(ini1=ini1,iter=iter,err=err,conver=conver))
}
```

```r
init <- seq(-10,20,0.5)
n <- length(init)
eps0 <- .Machine$double.eps^0.5
max_ite0 <- 200

val_NR <- rep(0,n)
conv_NR <- rep(0,n)
iter_NR <- rep(0,n)
time_NR <- rep(0,n)

for (i in 1:n){
  time_NR[i] <- system.time(res <- NR(init[i],samp,eps0,max_ite0))[1]
  val_NR[i] <- res$ini1
  iter_NR[i] <- res$iter
  conv_NR[i] <- res$conver
}

# time needed
time_NR
```

```
##  [1] 0.020 0.001 0.002 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001
## [12] 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001
## [23] 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.000 0.000 0.000
## [34] 0.001 0.001 0.001 0.001 0.001 0.000 0.001 0.000 0.000 0.001 0.001
## [45] 0.000 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001
## [56] 0.001 0.001 0.000 0.001 0.001 0.001
```

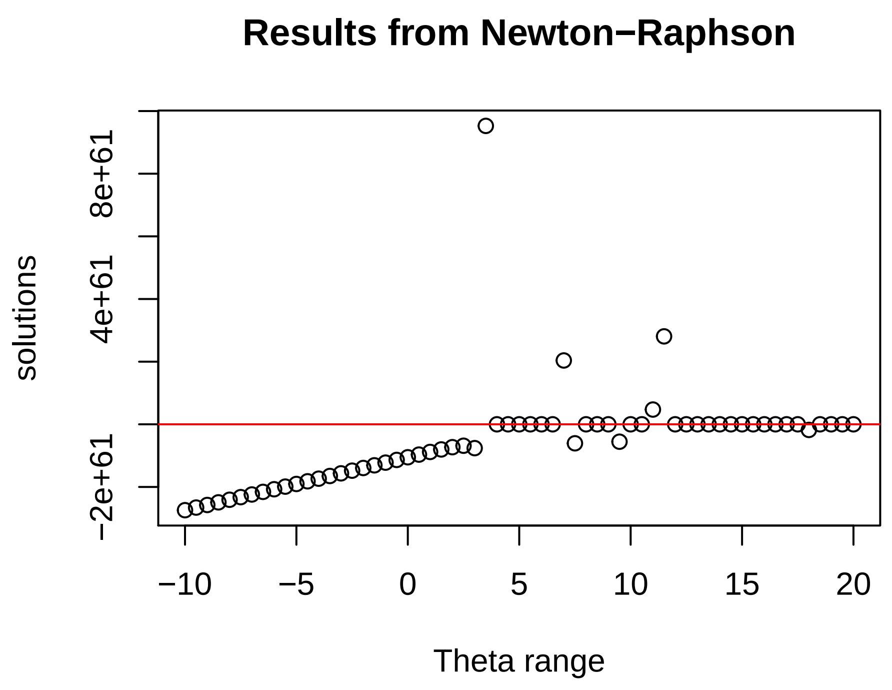
```r
# whether convergence
conv_NR
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1
## [36] 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```r
# iteration number
iter_NR
```

```
##  [1] 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
## [18] 200 200 200 200 200 200 200 200 200 200 200 200 200   6   4   5   8
## [35] 200 200 200 200   7 200 200   6 200 200 200 200 200 200 200 200 200
## [52] 200 200 200 200 200 200 200 200 200 200
```

```r
# Plot the results
plot(init, val_NR, xlab = "Theta range", ylab = "solutions",
     main = "Results from Newton-Raphson")
abline(h = 5, col = "red")
```



From the indicator vector conv_NR, we find that only 6 cased are convergent, which implies that the Newton-Raphson algorithm is really depend on the selection of initial value. When the initial value is closer to the root, NR method can give us a satisfying result within a small number of iterations. However, when the initial value is far from the root, the algorithm cannot converge.

3.3 Fixed-point method

```
FP <- function(ini,alph,x,tol,max_ite){
  err <- 100
  iter <- 0
  conver <- 0
  while ((err > tol) & (iter < max_ite)) {
    ini1 <- ini+alph*l1(ini,x)
    err <- abs(ini1-ini)
    ini <- ini1
    iter <- iter+1
  }
  if (iter >= max_ite) conver <- 1
  return(list(ini1=ini1,iter=iter,err=err,conver=conver))
}


alp <- c(1,0.64,0.25)
val_FP <- matrix(0,nrow=3,ncol=n)
iter_FP <- matrix(0,nrow=3,ncol=n)
conv_FP <- matrix(0,nrow=3,ncol=n)
time_FP <- matrix(0,nrow=3,ncol=n)
```

```r
for (i in 1:n){
  for (j in 1:3){
  time_FP[j,i] <- system.time(res <- FP(init[i],alp[j],samp,eps0,max_ite0))[1]
  val_FP[j,i] <- res$ini1
  iter_FP[j,i] <- res$iter
  conv_FP[j,i] <- res$conver
  }
}

# averaged time needed
apply(time_FP,1,mean)
```

```
## [1] 7.704918e-04 6.229508e-04 9.836066e-05
```

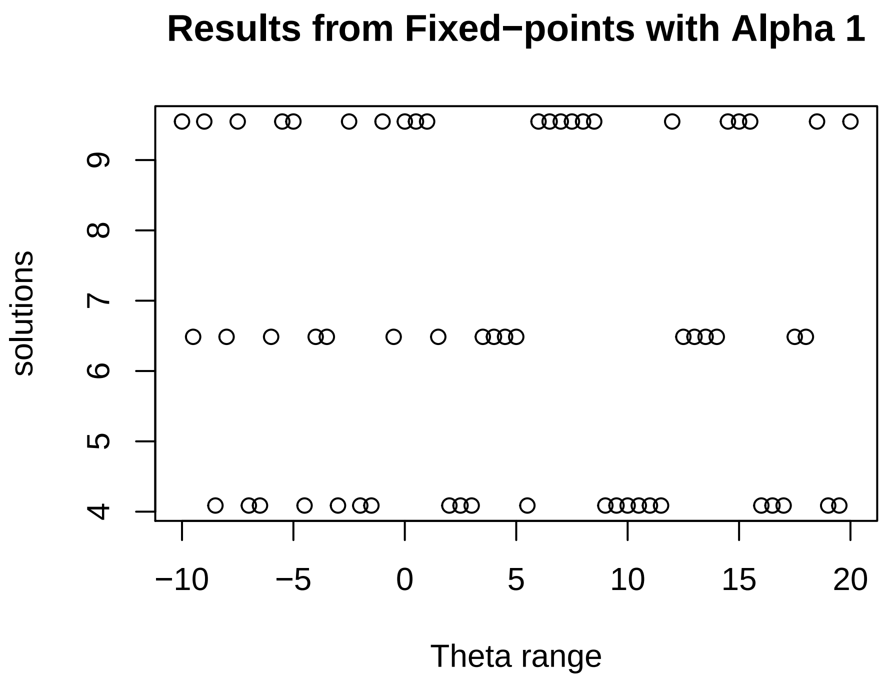
```r
# whether convergence
conv_FP
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [2,]    1    1    1    1    1    1    1    1    1     1     1     1     1
## [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]     1     1     1     1     1     1     1     1     1     1     1
## [2,]     1     1     1     1     1     1     1     1     1     1     1
## [3,]     0     0     0     0     0     0     0     0     0     0     0
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]     1     1     1     1     1     1     1     1     1     1     1
## [2,]     1     1     1     1     1     1     1     1     1     1     1
## [3,]     0     0     0     0     0     0     0     0     0     0     0
##      [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46]
## [1,]     1     1     1     1     1     1     1     1     1     1     1
## [2,]     1     1     1     1     1     1     1     1     1     1     1
## [3,]     0     0     0     0     0     0     0     0     0     0     0
##      [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57]
## [1,]     1     1     1     1     1     1     1     1     1     1     1
## [2,]     1     1     1     1     1     1     1     1     1     1     1
## [3,]     0     0     0     0     0     0     0     0     0     0     0
##      [,58] [,59] [,60] [,61]
## [1,]     1     1     1     1
## [2,]     1     1     1     1
## [3,]     0     0     0     0
```

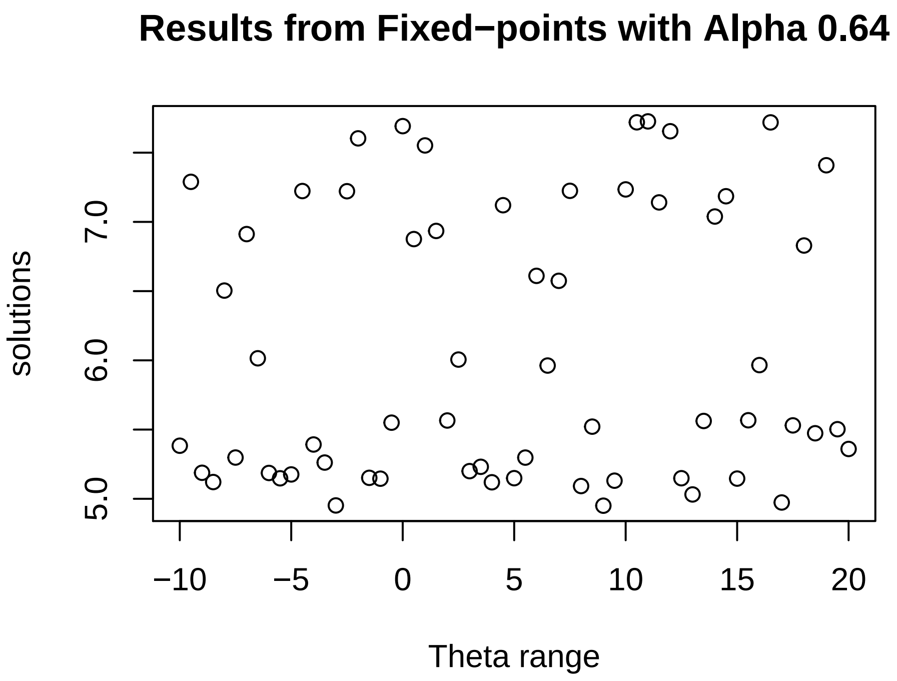
```r
# iteration number
iter_FP
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]  200  200  200  200  200  200  200  200  200   200   200   200   200
## [2,]  200  200  200  200  200  200  200  200  200   200   200   200   200
## [3,]   47   45   44   42   41   39   38   36   34    34    30    31    29
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]   200   200   200   200   200   200   200   200   200   200   200
## [2,]   200   200   200   200   200   200   200   200   200   200   200
## [3,]    28    27    26    25    25    24    23    23    21    20    21
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]   200   200   200   200   200   200   200   200   200   200   200
## [2,]   200   200   200   200   200   200   200   200   200   200   200
## [3,]    20    20    18    18    19    18    17    18    17    18    19
##      [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46]
## [1,]   200   200   200   200   200   200   200   200   200   200   200
## [2,]   200   200   200   200   200   200   200   200   200   200   200
## [3,]    19    20    21    19    22    23    22    24    20    25    25
##      [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57]
## [1,]   200   200   200   200   200   200   200   200   200   200   200
## [2,]   200   200   200   200   200   200   200   200   200   200   200
## [3,]    25    27    28    29    30    31    32    33    35    36    36
##      [,58] [,59] [,60] [,61]
## [1,]   200   200   200   200
## [2,]   200   200   200   200
## [3,]    40    43    46    50
```

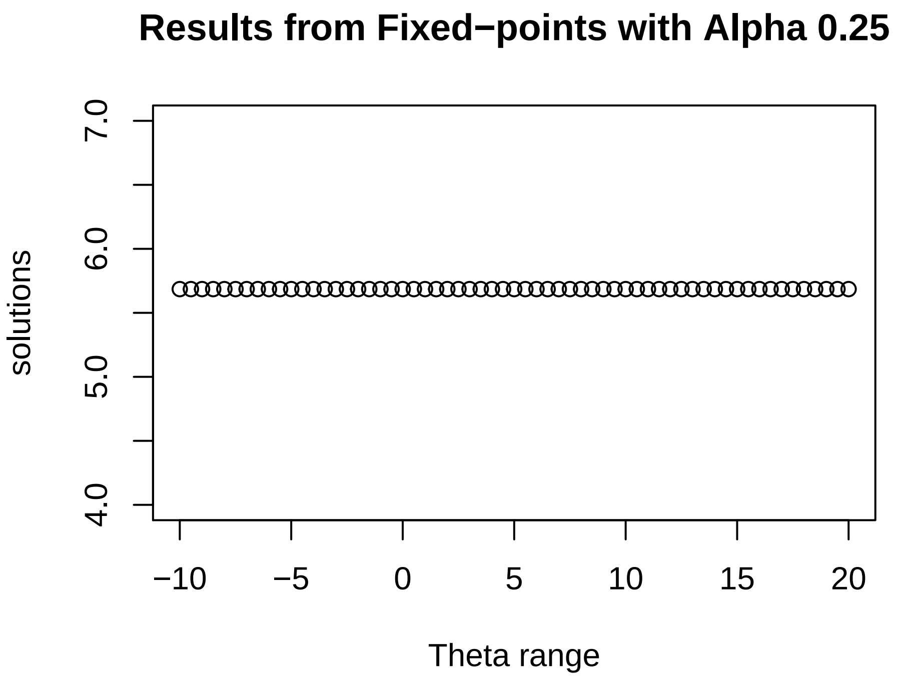
```
# Plot the results for alpha=1
plot(init, val_FP[1,], xlab = "Theta range", ylab = "solutions",
     main = "Results from Fixed-points with Alpha 1")
```



```
# Plot the results for alpha=0.64
plot(init, val_FP[2,], xlab = "Theta range", ylab = "solutions",
     main = "Results from Fixed-points with Alpha 0.64")
```



```r
# Plot the results for alpha=0.25
plot(init, val_FP[3,], ylim = c(4,7), xlab = "Theta range", ylab = "solutions",
     main = "Results from Fixed-points with Alpha 0.25")
```



The simulation considers the situation for different α's also different initial values. The selection of an appropriate α is crucial for the success of the algorithm. From the indicator matrix, we find that only when α is small enough, the algorithm converges and the solution is quite close to the root of the objective function. Otherwise the solution of the algorithm is far from the root.

3.4 Fisher-scoring method and Newton-Raphson method

```
FS <- function(ini,x,tol,max_ite){
  err <- 100
  iter <- 0
  conver <- 0
  while ((err > tol) & (iter < max_ite)) {
    ini1 <- ini+2*l1(ini,x)/length(x)
    err <- abs(ini1-ini)
    ini <- ini1
    iter <- iter+1
  }
  if (iter >= max_ite) conver <- 1
  return(list(ini1=ini1,iter=iter,err=err,conver=conver))
}

val_FS <- rep(0,n)
iter_FS <- rep(0,n)
conv_FS <- rep(0,n)
time_FS <- rep(0,n)

val_FSNR <- rep(0,n)
conv_FSNR <- rep(0,n)

for (i in 1:n){
  time_FS[i] <- system.time(res <- FS(init[i],samp,eps0,max_ite0))[1]
  val_FS[i] <- res$ini1
  iter_FS[i] <- res$iter
  conv_FS[i] <- res$conver
  # refine the estimate only when FS converges
  if (res$conver==0){
    res1 <- NR(res$ini1,samp,eps0,max_ite0)
    val_FSNR[i] <- res1$ini1
    conv_FSNR[i] <- res1$conver
  }else{
    val_FSNR[i] <- NA
    conv_FSNR[i] <- NA
  }
}

# time needed for Fisher scoring
time_FS
```

```
##  [1] 0.009 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.001 0.000
## [12] 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.000 0.000 0.000
## [23] 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.000 0.000 0.000 0.000
## [34] 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.001 0.000 0.000 0.000
## [45] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
## [56] 0.000 0.000 0.000 0.001 0.001 0.000
```

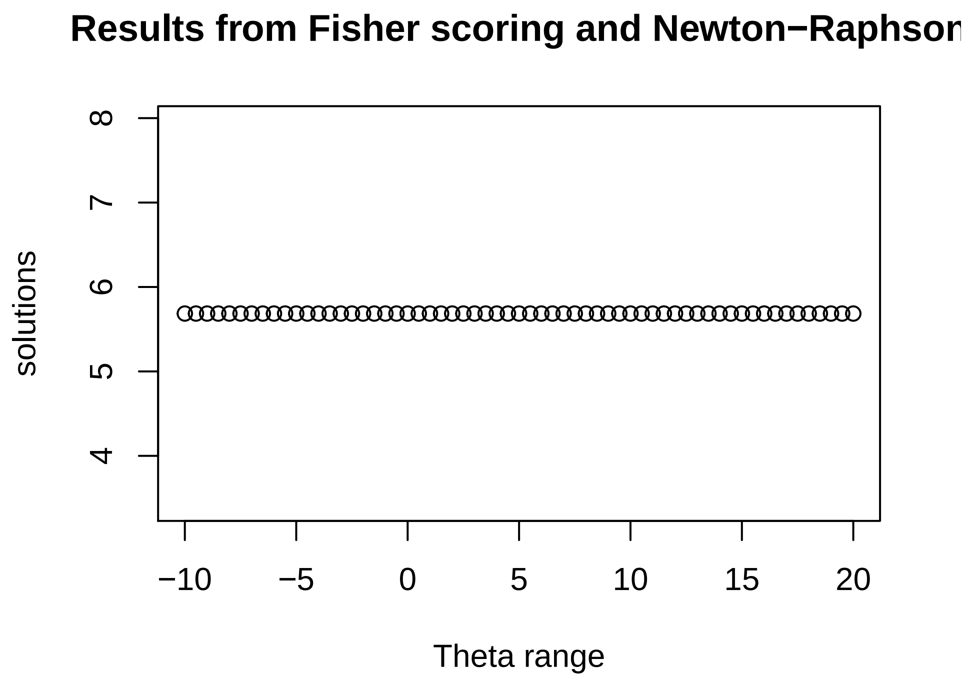
```
# whether convergence
conv_FS
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
# iteration number
iter_FS
```

```
##  [1] 45 42 40 38 36 35 33 31 29 28 27 25 23 22 21 19 19 17 16 15 15 13 12
## [24] 12 12 10 10  9 10  9  8  8  8  9 10 11 11 12 13 14 15 16 15 17 18 19
## [47] 19 20 21 23 24 25 27 28 30 32 34 37 40 44 49
```

```
# Plot the results
plot(init, val_FSNR, xlab = "Theta range", ylab = "solutions",
     main = "Results from Fisher scoring and Newton-Raphson")
```



The combination of Fisher scoring and Newton-Raphson can greatly improve the performance of Newton-Raphson. At first, we can use Fisher Scoring to provide a good initial value for Newton-Raphson, then the NR algorithm can refine the estimate. Also, for Fisher Scoring, when the initial value is closer to the root, the number of iterations needed to converge is smaller.

3.5 Comments on the results from different methods

In general, when the initial value is closer to the root, Newton-Raphson can attain that point with the fastest speed. But it is unstable to the initial value. For Fixed-point method, once we have a small enough α, even though the initial value is far from the root, it can converge to a value closer to the root. It is robust to the initial value. As for Fisher scoring, when we consider the estimation of a location parameter, the Fisher information will be a constant, which can facilitate the algorithm a lot without evaluating the Hessian matrix at every iteration. A combination of Fisher scoring and Newton-Raphson is a good choice.