Homework 3 - STAT 5362 Statistical Computing

Sen Yang[*]

22 September 2018

Abstract

This homework starts with a mathmetical proof of loglikelihood function and corresponding Fisher information of Cauchy distribution with a unknown scale parameter θ. Based on a random sample, I use Newton–Raphson method, Fixed-Point method and Fisher Scoring to estimate the value of θ. Finally, different methods are compared to each other, among which a combination of Fisher scoring and Newton-Raphson method has the most efficient way to do the estimation.

1 Proof in Mathematics

The density function of a Cauchy distribution with a known scale parameter is

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

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

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{n} \pi[1+(X_i-\theta)^2]^{-1} \\ &= \pi^{-n} \prod_{i=1}^{n} [1+(X_i-\theta)^2]^{-1}. \end{aligned}$$

Take logarithm on both sides,

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

Take first derivative of θ on both sides,

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

Take second derivative of θ on both sides,

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

[*] sen.2.yang@uconn.edu; M.S. student at Department of Statistics, University of Connecticut.

Then, the Fisher information $I_n(\theta)$ of this sample is

$$\begin{aligned}
I_n(\theta) &= -E_\theta[\ell''(\theta)|\theta] \\
&= \int_{-\infty}^{\infty} \ell''(\theta) f(x) dx \\
&= 2n \int_{-\infty}^{\infty} \frac{1-(\theta-x)^2}{[1+(x-\theta)^2]^2} \cdot \frac{1}{\pi[1+(x-\theta)^2]} dx \\
&= \frac{2n}{\pi} \int_{-\infty}^{\infty} \frac{1-(\theta-x)^2}{[1+(x-\theta)^2]^3} dx \\
&= \frac{2n}{\pi} \int_{-\infty}^{\infty} \frac{1-x^2}{[1+x^2]^3} dx \\
&= \frac{4n}{\pi} \int_{0}^{\infty} \frac{1-x^2}{[1+x^2]^3} dx.
\end{aligned}$$

Substituting $u = \frac{1}{1+x^2}$, then $x^2 = \frac{1}{u} - 1$ and $x = (\frac{1}{u} - 1)^{0.5} = (1-u)^{0.5} u^{-0.5}$.
Therefore, $dx = -0.5 \cdot [(1-u)^{-0.5} u^{-0.5} + (1-u)^{0.5} u^{-1.5}] du$.

$$\begin{aligned}
I_n(\theta) &= \frac{4n}{\pi} [\int_0^{\infty} \frac{1}{[1+x^2]^3} dx - \int_0^{\infty} \frac{x^2}{[1+x^2]^3} dx] \\
&= -\frac{2n}{\pi} \int_0^1 [u^{0.5}(1-u)^{1.5} - u^{2.5}(1-u)^{-0.5}] du \\
&= -\frac{2n}{\pi} [\int_0^1 u^{0.5}(1-u)^{1.5} du - \int_0^1 u^{2.5}(1-u)^{-0.5} du] \quad (\textit{Beta integral}) \\
&= -\frac{2n}{\pi} [\frac{\Gamma(1.5)\Gamma(2.5)}{\Gamma(4)} - \frac{\Gamma(3.5)\Gamma(0.5)}{\Gamma(4)}] \\
&= -\frac{2n}{\pi} [\frac{0.375\pi - 1.875\pi}{3!}] \\
&= \frac{n}{2}
\end{aligned}$$

2 Random Sample and Plot of Loglikelihood against θ

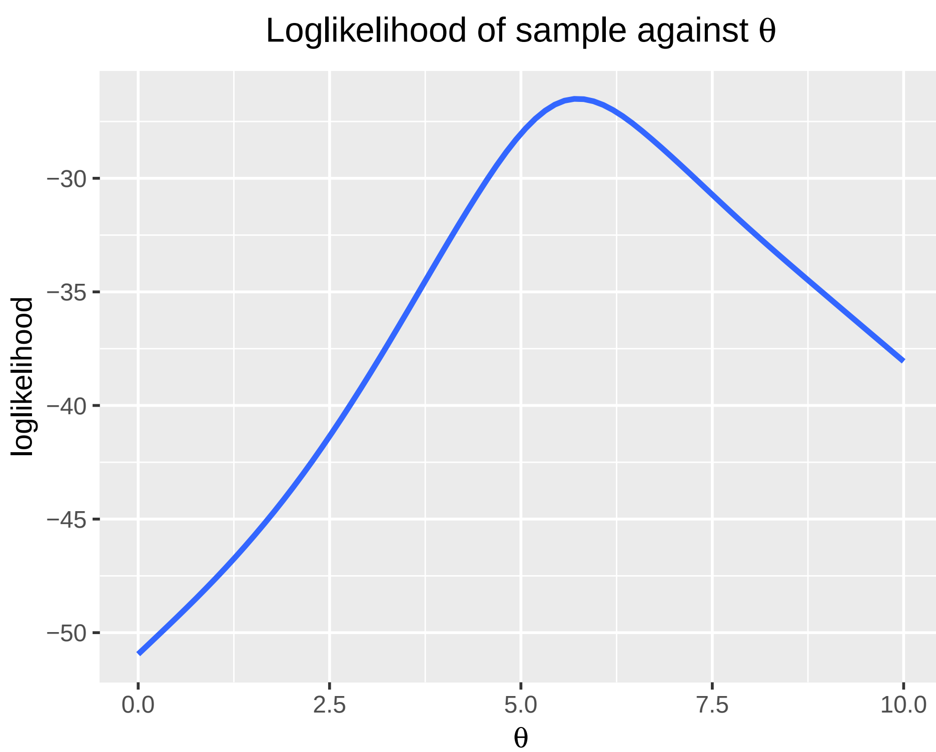
Set the random seed as 20180909 and generate a random sample of size $n = 10$ with $\theta = 5$.

```r
set.seed(20180909)
rdm_sample <- rcauchy(10, 5, 1)
rdm_sample
```

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

The plot of loglikelihood function against θ is shown in Figure **??**.

```r
log_llh <- function(theta, sample_X){
  lsum <- 0
  for (i in 1:length(sample_X)){
    lsum <- lsum + -log(pi) - log(1 + (theta - sample_X[i])^2)
  }
  lsum
}

x <- seq(0, 10, 0.01)
y <- sapply(x, log_llh, sample_X = rdm_sample)
data_l <- as.data.frame(cbind(y, x))
library(ggplot2)
ggplot(data_l, aes(x, y)) + geom_smooth() +
  labs(title = expression(paste("Loglikelihood of sample against ", theta)),
       x = expression(theta), y = "loglikelihood") +
  theme(plot.title = element_text(hjust = 0.5))
```



3 Newton–Raphson method

```r
#first derivative of loglikelihood
first_derv <- function(theta, sample_X) {
  first_derv <- 0
```

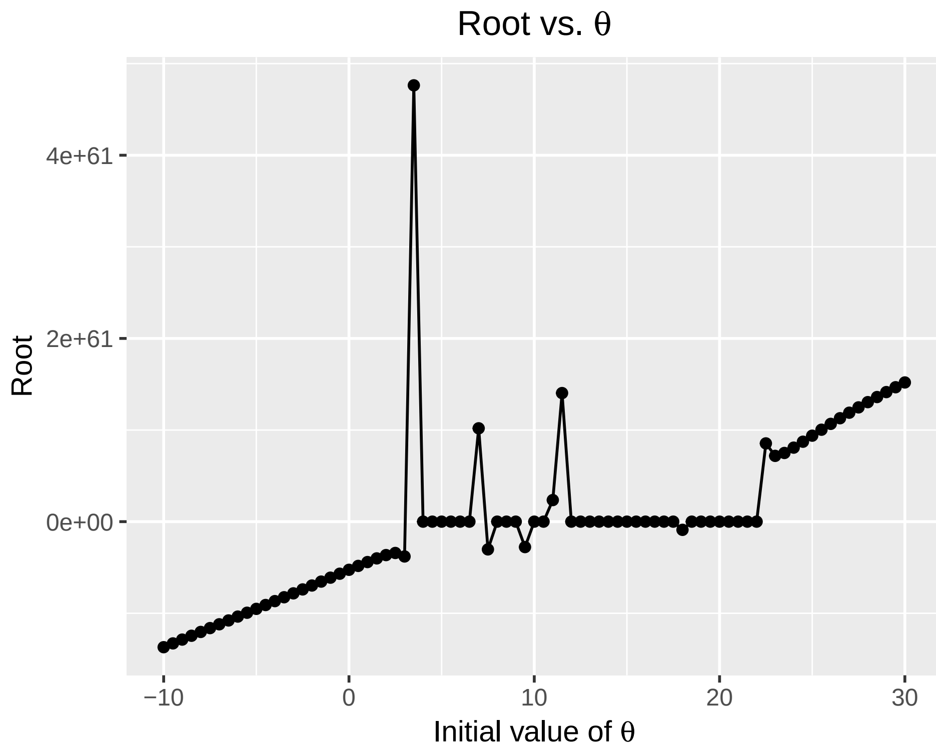
```r
  for (i in 1:length(sample_X)){
    first_derv <- first_derv -2 *
      ((theta - sample_X[i])/(1 + (theta - sample_X[i])^2))
  }
  first_derv
}
#second derivative of loglikelihood
second_derv <- function(theta, sample_X) {
  second_derv <- 0
  for (i in 1:length(sample_X)){
    second_derv <- second_derv -2 *
      ((1-(theta - sample_X[i])^2)/(1 + (theta - sample_X[i])^2)^2)
  }
  second_derv
}
#Newton-Raphson method
newton <- function(init, pre=1e-50, maxrun=200) {
  n <- 1
  xt <- init
  while (n<maxrun){
    fx <- first_derv(xt, rdm_sample)
    fx_d <- second_derv(xt, rdm_sample)
    if (fx == 0) {break}
    ht <- -fx/fx_d
    xt1 <- xt + ht
    if (abs(xt1-xt) < pre) {break}
    xt <- xt1
    n <- n+1
  }
return(c(root = xt, iter = n))
}
init <- seq(-10, 30, 0.5)
result <- as.data.frame(matrix(0, nrow = length(init), ncol = 3))
for (i in 1:length(init) ) {
  result[i,1] <- paste("Initial = ", init[i])
  result[i,2:3] <- newton(init[i])
}
colnames(result) <- (c("Initial", "Root", "# of iterations"))
library(pander)
pander(result)
```

Initial	Root	# of iterations
Initial = -10	-1.371e+61	200
Initial = -9.5	-1.329e+61	200
Initial = -9	-1.287e+61	200
Initial = -8.5	-1.246e+61	200

Initial	Root	# of iterations
Initial = -8	-1.204e+61	200
Initial = -7.5	-1.162e+61	200
Initial = -7	-1.12e+61	200
Initial = -6.5	-1.078e+61	200
Initial = -6	-1.036e+61	200
Initial = -5.5	-9.943e+60	200
Initial = -5	-9.521e+60	200
Initial = -4.5	-9.099e+60	200
Initial = -4	-8.676e+60	200
Initial = -3.5	-8.251e+60	200
Initial = -3	-7.826e+60	200
Initial = -2.5	-7.399e+60	200
Initial = -2	-6.972e+60	200
Initial = -1.5	-6.544e+60	200
Initial = -1	-6.115e+60	200
Initial = -0.5	-5.686e+60	200
Initial = 0	-5.257e+60	200
Initial = 0.5	-4.832e+60	200
Initial = 1	-4.413e+60	200
Initial = 1.5	-4.01e+60	200
Initial = 2	-3.651e+60	200
Initial = 2.5	-3.421e+60	200
Initial = 3	-3.802e+60	200
Initial = 3.5	4.763e+61	200
Initial = 4	21.08	200
Initial = 4.5	19.38	200
Initial = 5	5.685	7
Initial = 5.5	5.685	5
Initial = 6	5.685	6
Initial = 6.5	5.685	9
Initial = 7	1.019e+61	200
Initial = 7.5	-3.029e+60	200
Initial = 8	20.56	200
Initial = 8.5	19.38	200
Initial = 9	5.685	8
Initial = 9.5	-2.776e+60	200
Initial = 10	19.38	200
Initial = 10.5	5.685	7
Initial = 11	2.358e+60	200
Initial = 11.5	1.404e+61	200
Initial = 12	21.08	200
Initial = 12.5	21.08	200
Initial = 13	21.08	200
Initial = 13.5	19.38	200
Initial = 14	19.38	200
Initial = 14.5	19.38	200

Initial	Root	# of iterations
Initial = 15	20.56	200
Initial = 15.5	21.08	200
Initial = 16	21.08	200
Initial = 16.5	20.56	200
Initial = 17	20.56	200
Initial = 17.5	20.56	200
Initial = 18	-8.954e+59	200
Initial = 18.5	21.08	200
Initial = 19	21.08	200
Initial = 19.5	21.08	200
Initial = 20	21.08	200
Initial = 20.5	19.38	200
Initial = 21	20.56	200
Initial = 21.5	20.56	200
Initial = 22	19.38	200
Initial = 22.5	8.543e+60	200
Initial = 23	7.181e+60	200
Initial = 23.5	7.498e+60	200
Initial = 24	8.079e+60	200
Initial = 24.5	8.727e+60	200
Initial = 25	9.386e+60	200
Initial = 25.5	1.004e+61	200
Initial = 26	1.067e+61	200
Initial = 26.5	1.129e+61	200
Initial = 27	1.189e+61	200
Initial = 27.5	1.248e+61	200
Initial = 28	1.305e+61	200
Initial = 28.5	1.36e+61	200
Initial = 29	1.414e+61	200
Initial = 29.5	1.467e+61	200
Initial = 30	1.519e+61	200

```r
# plot
result <- cbind(init, result)
ggplot(result, aes(init, Root)) +
  geom_line() + geom_point() +
  labs(title = expression(paste("Root vs. ", theta)),
       x = expression(paste("Initial value of ", theta)), y = "Root") +
  theme(plot.title = element_text(hjust = 0.5))
```



According to the data and figure, it can be concluded that when initial value is less then 4, the estimation is not accurate. When it is around or a little bit larger than 5, the estimation is very close to the true value. Meanwhile, at some initial points, the estimations are not stable.

4 Improved Newton–Raphson method

By halving the steps, an improved Newton-Raphson method is used to do the estimation.

```
# Improve it by halving the steps
# Improved Newton-Raphson method
newton2 <- function(init, pre=1e-50, maxrun=200) {
  n <- 1
  xt <- init
  while (n<maxrun){
    fx <- first_derv(xt, rdm_sample)
    fx_d <- second_derv(xt, rdm_sample)
    if (fx == 0) {break}
    ht <- -fx/fx_d
    xt1 <- xt + ht/2
    if (abs(xt1-xt) < pre) {break}
    xt <- xt1
    n <- n+1
  }
  return(c(root = xt, iter = n))
```

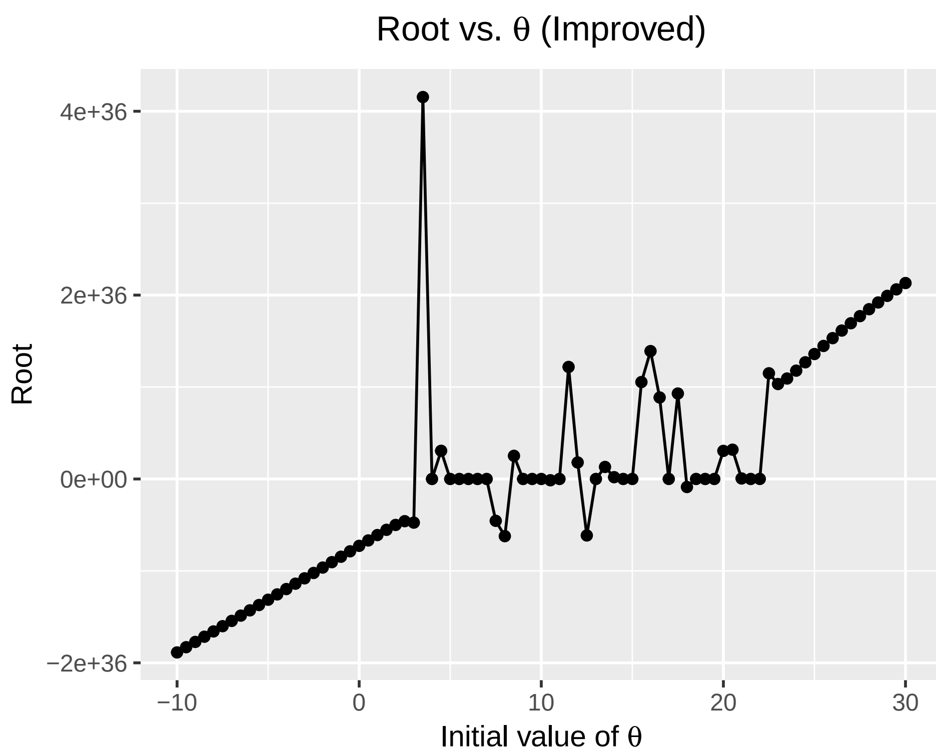
```
}

init <- seq(-10, 30, 0.5)
result2 <- as.data.frame(matrix(0, nrow = length(init), ncol = 3))
for (i in 1:length(init) ) {
  result2[i,1] <- paste("Initial = ", init[i])
  result2[i,2:3] <- newton2(init[i])
}
colnames(result2) <- (c("Initial", "Root", "# of iterations"))
library(pander)
pander(result2)
```

Initial	Root	# of iterations
Initial = -10	-1.887e+36	200
Initial = -9.5	-1.83e+36	200
Initial = -9	-1.773e+36	200
Initial = -8.5	-1.716e+36	200
Initial = -8	-1.659e+36	200
Initial = -7.5	-1.601e+36	200
Initial = -7	-1.544e+36	200
Initial = -6.5	-1.486e+36	200
Initial = -6	-1.429e+36	200
Initial = -5.5	-1.371e+36	200
Initial = -5	-1.313e+36	200
Initial = -4.5	-1.255e+36	200
Initial = -4	-1.197e+36	200
Initial = -3.5	-1.139e+36	200
Initial = -3	-1.081e+36	200
Initial = -2.5	-1.022e+36	200
Initial = -2	-9.636e+35	200
Initial = -1.5	-9.048e+35	200
Initial = -1	-8.458e+35	200
Initial = -0.5	-7.867e+35	200
Initial = 0	-7.275e+35	200
Initial = 0.5	-6.686e+35	200
Initial = 1	-6.101e+35	200
Initial = 1.5	-5.532e+35	200
Initial = 2	-5.002e+35	200
Initial = 2.5	-4.593e+35	200
Initial = 3	-4.748e+35	200
Initial = 3.5	4.155e+36	200
Initial = 4	3.098e+22	200
Initial = 4.5	3.065e+35	200
Initial = 5	5.685	51
Initial = 5.5	5.685	49
Initial = 6	5.685	49

Initial	Root	# of iterations
Initial = 6.5	5.685	43
Initial = 7	5.685	52
Initial = 7.5	-4.555e+35	200
Initial = 8	-6.219e+35	200
Initial = 8.5	2.518e+35	200
Initial = 9	5.685	52
Initial = 9.5	5.685	54
Initial = 10	3.672e+30	200
Initial = 10.5	-1.404e+34	200
Initial = 11	5.685	53
Initial = 11.5	1.219e+36	200
Initial = 12	1.796e+35	200
Initial = 12.5	-6.143e+35	200
Initial = 13	-2.702e+31	200
Initial = 13.5	1.309e+35	200
Initial = 14	1.952e+34	200
Initial = 14.5	9.968e+31	200
Initial = 15	6.606e+31	200
Initial = 15.5	1.054e+36	200
Initial = 16	1.391e+36	200
Initial = 16.5	8.864e+35	200
Initial = 17	2.746e+32	200
Initial = 17.5	9.295e+35	200
Initial = 18	-8.746e+34	200
Initial = 18.5	9.625e+28	200
Initial = 19	2.95e+30	200
Initial = 19.5	1.306e+29	200
Initial = 20	3.055e+35	200
Initial = 20.5	3.186e+35	200
Initial = 21	5.516e+33	200
Initial = 21.5	-2.126e+30	200
Initial = 22	7.038e+32	200
Initial = 22.5	1.15e+36	200
Initial = 23	1.033e+36	200
Initial = 23.5	1.093e+36	200
Initial = 24	1.178e+36	200
Initial = 24.5	1.269e+36	200
Initial = 25	1.359e+36	200
Initial = 25.5	1.447e+36	200
Initial = 26	1.532e+36	200
Initial = 26.5	1.614e+36	200
Initial = 27	1.694e+36	200
Initial = 27.5	1.771e+36	200
Initial = 28	1.847e+36	200
Initial = 28.5	1.92e+36	200
Initial = 29	1.992e+36	200

Initial	Root	# of iterations
Initial = 29.5	2.062e+36	200
Initial = 30	2.131e+36	200

```r
# plot
result2 <- cbind(init, result2)
ggplot(result2, aes(init, Root)) +
  geom_line() + geom_point() +
  labs(title = expression(paste("Root vs. ", theta, " (Improved)")),
       x = expression(paste("Initial value of ", theta)), y = "Root") +
  theme(plot.title = element_text(hjust = 0.5))
```



Compared to the standard Newton-Raphson method, the improved one has a more narrow estimation range. However, obviously on the plot, the estimation is still not stable.

5 Fixed-Point Iterations

```r
fix_pnt <- function(init, alpha, pre=1e-50, maxrun=200) {
  n <- 1
  x <- init
  while (n<maxrun){
```

```r
    fx <- first_derv(x, rdm_sample)
    if (fx == 0) {break}
    Gx <- x + alpha*fx
    if (abs(Gx-x) < pre) {break}
    x <- Gx
    n <- n+1
  }
  return(c(root = x, iter = n))
}

init <- seq(-10, 30, 0.5)
alpha <- c(1, 0.64, 0.25)
result3 <- as.data.frame(matrix(0, nrow = length(init), ncol = 7))
for (i in 1:length(init) ) {
  result3[i,1] <- paste("Init.=", init[i])
  result3[i,2:3] <- fix_pnt(init[i], alpha[1])
  result3[i,4:5] <- fix_pnt(init[i], alpha[2])
  result3[i,6:7] <- fix_pnt(init[i], alpha[3])
}
colnames(result3) <- c("Initial", paste("Root (alpha=",alpha[1],")"),
                        paste0("# of iterations (alpha=",alpha[1],")"),
                        paste0("Root (alpha=",alpha[2],")"),
                        paste0("# of iterations (alpha=",alpha[2],")"),
                        paste0("Root (alpha=",alpha[3],")"),
                        paste0("# of iterations (alpha=",alpha[3],")") )
library(pander)
pander(result3, style="rmarkdown", split.table=Inf, split.cells=Inf)
```

Initial	Root (alpha= 1)	# of iterations (alpha=1)	Root (alpha=0.64)	# of iterations (alpha=0.64)	Root (alpha=0.25)	# of iterations (alpha=0.25)
Init.= -10	4.087	200	7.454	200	5.685	65
Init.= -9.5	9.548	200	5.151	200	5.685	62
Init.= -9	4.087	200	5.991	200	5.685	60
Init.= -8.5	6.486	200	6.991	200	5.685	60
Init.= -8	9.548	200	5.411	200	5.685	59
Init.= -7.5	4.087	200	7.396	200	5.685	57
Init.= -7	6.486	200	5.285	200	5.685	56
Init.= -6.5	6.486	200	5.309	200	5.685	54

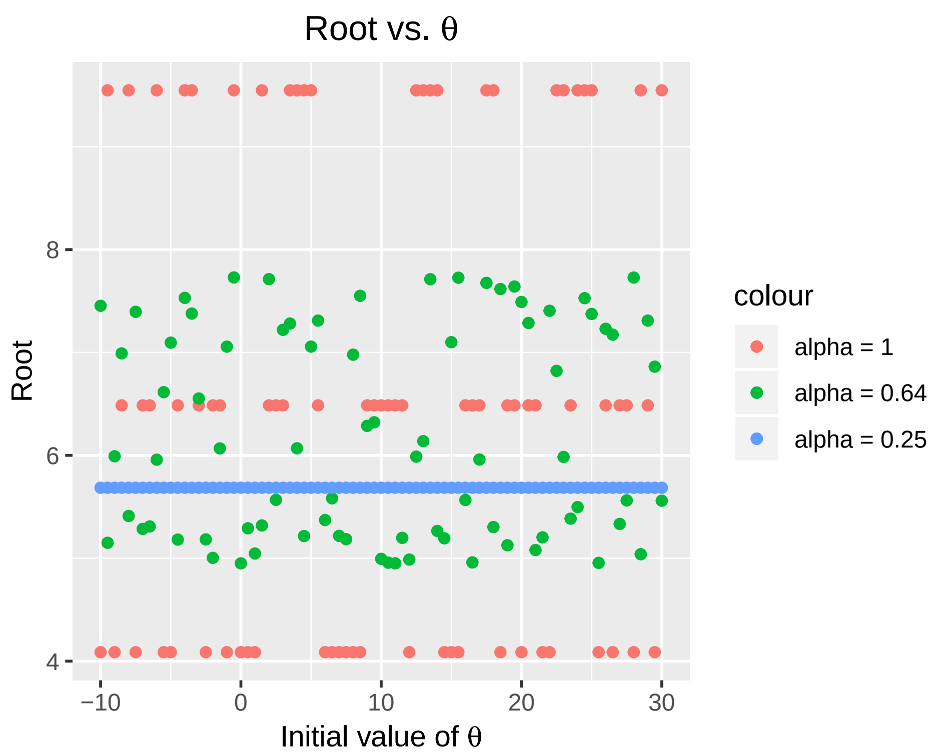
Initial	Root (alpha= 1)	# of iterations (alpha=1)	Root (alpha=0.64)	# of iterations (alpha=0.64)	Root (alpha=0.25)	# of iterations (alpha=0.25)
Init.= -6	9.548	200	5.958	200	5.685	51
Init.= -5.5	4.087	200	6.614	200	5.685	52
Init.= -5	4.087	200	7.096	200	5.685	48
Init.= -4.5	6.486	200	5.182	200	5.685	49
Init.= -4	9.548	200	7.531	200	5.685	46
Init.= -3.5	9.548	200	7.378	200	5.685	45
Init.= -3	6.486	200	6.553	200	5.685	44
Init.= -2.5	4.087	200	5.183	200	5.685	43
Init.= -2	6.486	200	5.004	200	5.685	42
Init.= -1.5	6.486	200	6.068	200	5.685	41
Init.= -1	4.087	200	7.057	200	5.685	42
Init.= -0.5	9.548	200	7.729	200	5.685	40
Init.= 0	4.087	200	4.951	200	5.685	41
Init.= 0.5	4.087	200	5.291	200	5.685	38
Init.= 1	4.087	200	5.046	200	5.685	36
Init.= 1.5	9.548	200	5.319	200	5.685	37
Init.= 2	6.486	200	7.713	200	5.685	38
Init.= 2.5	6.486	200	5.568	200	5.685	36
Init.= 3	6.486	200	7.22	200	5.685	35
Init.= 3.5	9.548	200	7.281	200	5.685	35
Init.= 4	9.548	200	6.069	200	5.685	37
Init.= 4.5	9.548	200	5.216	200	5.685	36

Initial	Root (alpha= 1)	# of iterations (alpha=1)	Root (alpha=0.64)	# of iterations (alpha=0.64)	Root (alpha=0.25)	# of iterations (alpha=0.25)
Init.= 5	9.548	200	7.057	200	5.685	34
Init.= 5.5	6.486	200	7.31	200	5.685	34
Init.= 6	4.087	200	5.372	200	5.685	33
Init.= 6.5	4.087	200	5.583	200	5.685	36
Init.= 7	4.087	200	5.218	200	5.685	35
Init.= 7.5	4.087	200	5.185	200	5.685	35
Init.= 8	4.087	200	6.979	200	5.685	38
Init.= 8.5	4.087	200	7.551	200	5.685	37
Init.= 9	6.486	200	6.287	200	5.685	35
Init.= 9.5	6.486	200	6.322	200	5.685	40
Init.= 10	6.486	200	4.995	200	5.685	39
Init.= 10.5	6.486	200	4.957	200	5.685	39
Init.= 11	6.486	200	4.951	200	5.685	42
Init.= 11.5	6.486	200	5.199	200	5.685	37
Init.= 12	4.087	200	4.988	200	5.685	43
Init.= 12.5	9.548	200	5.988	200	5.685	42
Init.= 13	9.548	200	6.138	200	5.685	42
Init.= 13.5	9.548	200	7.712	200	5.685	45
Init.= 14	9.548	200	5.265	200	5.685	46
Init.= 14.5	4.087	200	5.194	200	5.685	47
Init.= 15	4.087	200	7.1	200	5.685	48
Init.= 15.5	4.087	200	7.727	200	5.685	49

Initial	Root (alpha= 1)	# of iterations (alpha=1)	Root (alpha=0.64)	# of iterations (alpha=0.64)	Root (alpha=0.25)	# of iterations (alpha=0.25)
Init.= 16	6.486	200	5.567	200	5.685	48
Init.= 16.5	6.486	200	4.96	200	5.685	50
Init.= 17	6.486	200	5.96	200	5.685	53
Init.= 17.5	9.548	200	7.676	200	5.685	53
Init.= 18	9.548	200	5.303	200	5.685	53
Init.= 18.5	4.087	200	7.617	200	5.685	57
Init.= 19	6.486	200	5.127	200	5.685	61
Init.= 19.5	6.486	200	7.641	200	5.685	64
Init.= 20	4.087	200	7.491	200	5.685	67
Init.= 20.5	6.486	200	7.287	200	5.685	72
Init.= 21	6.486	200	5.08	200	5.685	78
Init.= 21.5	4.087	200	5.203	200	5.685	78
Init.= 22	4.087	200	7.406	200	5.685	77
Init.= 22.5	9.548	200	6.821	200	5.685	79
Init.= 23	9.548	200	5.985	200	5.685	80
Init.= 23.5	6.486	200	5.385	200	5.685	82
Init.= 24	9.548	200	5.497	200	5.685	83
Init.= 24.5	9.548	200	7.528	200	5.685	86
Init.= 25	9.548	200	7.374	200	5.685	85
Init.= 25.5	4.087	200	4.956	200	5.685	87
Init.= 26	6.486	200	7.23	200	5.685	90
Init.= 26.5	4.087	200	7.173	200	5.685	90

Initial	Root (alpha= 1)	# of iterations (alpha=1)	Root (alpha=0.64)	# of iterations (alpha=0.64)	Root (alpha=0.25)	# of iterations (alpha=0.25)
Init.= 27	6.486	200	5.334	200	5.685	91
Init.= 27.5	6.486	200	5.562	200	5.685	95
Init.= 28	4.087	200	7.728	200	5.685	95
Init.= 28.5	9.548	200	5.039	200	5.685	97
Init.= 29	6.486	200	7.31	200	5.685	98
Init.= 29.5	4.087	200	6.862	200	5.685	100
Init.= 30	9.548	200	5.56	200	5.685	102

```r
# plot
result3_plot <- cbind(init, result3)
colnames(result3_plot)[c(3,5,7)] <- c("y1","y2","y3")
ggplot(result3_plot, aes(init)) +
  geom_point(aes(y = y1, colour = "var0")) +
  geom_point(aes(y = y2, colour = "var1")) +
  geom_point(aes(y = y3, colour = "var2")) +
  labs(title = expression(paste("Root vs. ", theta)),
       x = expression(paste("Initial value of ", theta)), y = "Root") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_colour_discrete(breaks=c("var0", "var1", "var2"),
                        labels=c("alpha = 1", "alpha = 0.64", "alpha = 0.25"))
```



According to the plot, by using 3 different α, we get 3 estimations with different performance. The smaller the α is, the faster and accurate the estimation is converging.

6 Fisher Scoring and Newton-Raphson method

```
# Fisher Scoring
fisher <- function(init, pre=1e-10, maxrun=200) {
  n <- 1
  Ix <- 10/2
  xt <- init
  while (n<maxrun){
    fx <- first_derv(xt, rdm_sample)
    if (fx == 0) {break}
    xt1 <- xt + fx/Ix
    if (abs(xt1-xt) < pre) {break}
    xt <- xt1
    n <- n+1
  }
  return(c(root = xt, iter = n))
}

init <- seq(-10, 30, 0.5)
result4 <- as.data.frame(matrix(0, nrow = length(init), ncol = 5))
```

```r
options(digits = 8)
for (i in 1:length(init) ) {
  result4[i,1] <- paste("Initial = ", init[i])
  result4[i,2:3] <- fisher(init[i])
  result4[i,4:5] <- newton(result4[i,2])
}
colnames(result4) <- c("Initial", "Root(Fisher Scoring)",
                       "# of iterations(Fisher Scoring)",
                       "Root(Newton-Raphson)",
                       "# of iterations (Newton-Raphson)")
library(pander)
pander(result4, style="rmarkdown",split.table=Inf, split.cells=Inf)
```

Initial	Root(Fisher Scoring)	# of iterations(Fisher Scoring)	Root(Newton-Raphson)	# of iterations (Newton-Raphson)
Initial = -10	5.685	47	5.685	2
Initial = -9.5	5.685	44	5.685	2
Initial = -9	5.685	42	5.685	2
Initial = -8.5	5.685	40	5.685	2
Initial = -8	5.685	39	5.685	2
Initial = -7.5	5.685	37	5.685	2
Initial = -7	5.685	36	5.685	2
Initial = -6.5	5.685	33	5.685	2
Initial = -6	5.685	31	5.685	2
Initial = -5.5	5.685	30	5.685	2
Initial = -5	5.685	29	5.685	2
Initial = -4.5	5.685	27	5.685	2
Initial = -4	5.685	25	5.685	2
Initial = -3.5	5.685	24	5.685	2
Initial = -3	5.685	23	5.685	2

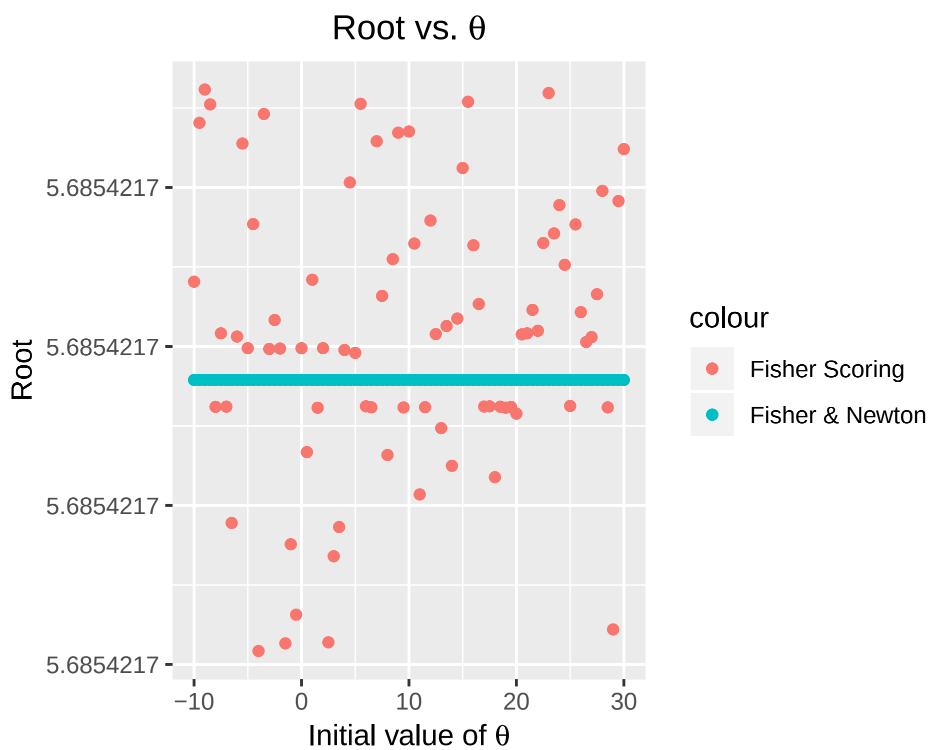
Initial	Root(Fisher Scoring)	# of iterations(Fisher Scoring)	Root(Newton-Raphson)	# of iterations (Newton-Raphson)
Initial = -2.5	5.685	21	5.685	2
Initial = -2	5.685	21	5.685	2
Initial = -1.5	5.685	19	5.685	2
Initial = -1	5.685	18	5.685	2
Initial = -0.5	5.685	17	5.685	2
Initial = 0	5.685	17	5.685	2
Initial = 0.5	5.685	15	5.685	2
Initial = 1	5.685	14	5.685	2
Initial = 1.5	5.685	15	5.685	2
Initial = 2	5.685	14	5.685	2
Initial = 2.5	5.685	12	5.685	2
Initial = 3	5.685	12	5.685	2
Initial = 3.5	5.685	11	5.685	2
Initial = 4	5.685	12	5.685	2
Initial = 4.5	5.685	11	5.685	2
Initial = 5	5.685	11	5.685	2
Initial = 5.5	5.685	10	5.685	2
Initial = 6	5.685	11	5.685	2
Initial = 6.5	5.685	12	5.685	2
Initial = 7	5.685	12	5.685	2
Initial = 7.5	5.685	13	5.685	2
Initial = 8	5.685	13	5.685	2
Initial = 8.5	5.685	14	5.685	2
Initial = 9	5.685	15	5.685	2
Initial = 9.5	5.685	17	5.685	2
Initial = 10	5.685	17	5.685	2
Initial = 10.5	5.685	18	5.685	2

Initial	Root(Fisher Scoring)	# of iterations(Fisher Scoring)	Root(Newton-Raphson)	# of iterations (Newton-Raphson)
Initial = 11	5.685	17	5.685	2
Initial = 11.5	5.685	20	5.685	2
Initial = 12	5.685	20	5.685	2
Initial = 12.5	5.685	21	5.685	2
Initial = 13	5.685	21	5.685	2
Initial = 13.5	5.685	22	5.685	2
Initial = 14	5.685	23	5.685	2
Initial = 14.5	5.685	25	5.685	2
Initial = 15	5.685	26	5.685	2
Initial = 15.5	5.685	27	5.685	2
Initial = 16	5.685	29	5.685	2
Initial = 16.5	5.685	30	5.685	2
Initial = 17	5.685	33	5.685	2
Initial = 17.5	5.685	35	5.685	2
Initial = 18	5.685	36	5.685	2
Initial = 18.5	5.685	40	5.685	2
Initial = 19	5.685	43	5.685	2
Initial = 19.5	5.685	47	5.685	2
Initial = 20	5.685	51	5.685	2
Initial = 20.5	5.685	59	5.685	2
Initial = 21	5.685	64	5.685	2
Initial = 21.5	5.685	65	5.685	2

Initial	Root(Fisher Scoring)	# of iterations(Fisher Scoring)	Root(Newton-Raphson)	# of iterations (Newton-Raphson)
Initial = 22	5.685	67	5.685	2
Initial = 22.5	5.685	68	5.685	2
Initial = 23	5.685	69	5.685	2
Initial = 23.5	5.685	70	5.685	2
Initial = 24	5.685	72	5.685	2
Initial = 24.5	5.685	73	5.685	2
Initial = 25	5.685	76	5.685	2
Initial = 25.5	5.685	77	5.685	2
Initial = 26	5.685	79	5.685	2
Initial = 26.5	5.685	81	5.685	2
Initial = 27	5.685	83	5.685	2
Initial = 27.5	5.685	85	5.685	2
Initial = 28	5.685	87	5.685	2
Initial = 28.5	5.685	90	5.685	2
Initial = 29	5.685	90	5.685	2
Initial = 29.5	5.685	94	5.685	2
Initial = 30	5.685	96	5.685	2

```r
# plot
# plot
result4_plot <- cbind(init, result4)
colnames(result4_plot)[c(3,5)] <- c("y1","y2")
ggplot(result4_plot, aes(init)) +
  geom_point(aes(y = y1, colour = "var0")) +
  geom_point(aes(y = y2, colour = "var1")) +
  labs(title = expression(paste("Root vs. ", theta)),
```

```r
      x = expression(paste("Initial value of ", theta)), y = "Root") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_colour_discrete(breaks=c("var0", "var1"),
                        labels=c("Fisher Scoring", "Fisher & Newton"))
```



According to the plot, it is obvious that the combined method of refining the Fisher Scoring estimate by using Newton-Raphson method is much better.

7 Comments

By the comparision of these several methods, we can conclude when we have a initial value around the true value, the Newton-Raphson method would give a good estimation. Otherwise, this method shows to be very unstable. For fixed-point method, we need a relatively small parameter α to get a decent estimation. By combining Newton-Raphson method and Fisher scoring method, we finally get a stable, accurate algorithm to do the estimation. In addition, when paying attention to the number of iterations, we can find the last method is also the fastest one.