Jared Yu

Computational Statistics

Problem Set 2

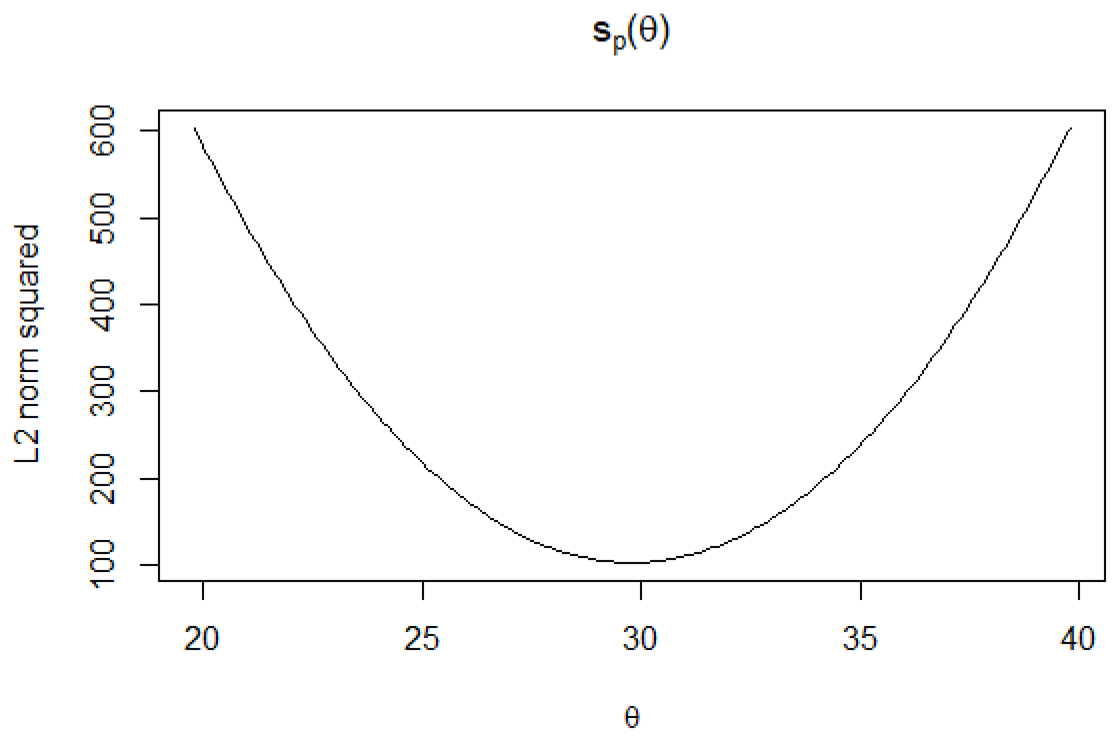
Note: All code is included in the appendix at the end of the document.

1. The following data are an i.i.d. sample from a $Normal(\theta, 1)$ distribution: 28, 33, 22, 35, 31. We wish to estimate θ by minimizing residuals. We will use the L_2 norm squared as our metric.
   a. What is the function $s_p(\theta)$ that we wish to minimize?

$$s_p(\theta) = \sum_{i=1}^{n} |y_i - g(\theta)|^p \rightarrow \sum_{i=1}^{n} (y_i - \theta)^2$$

   b. Graph $s_p(\theta)$.

Since the value θ is unknown and the variance is known to be 1, then θ will be estimated using MLE. Taking the derivative of the natural log of a Normal distribution with parameters $(\theta, 1)$ and setting it to 0 leads to $\hat{\theta} = \bar{y}$. Here, the sample mean is 29.8 and so in the graph below it's clear that the minimum of the graph is around that area.



   c. Find the Minimum Residual Estimator for θ using the Bisection Method correct to 2 decimal places.

In trying to minimize the objective function, the Bisection method can determine that the value of θ is minimized at $\theta = 29.79492$ in 9 iterations. This rounds off to 29.80, which is the same as the MLE as determined using the analytical method.

d. If we were to use Newton's Method to solve this optimization problem, what would the refinement increment $h(t)$ be?

$$s_2(\theta) = \sum_{i=1}^{n} (y_i - \theta)^2$$

$$s_2'(\theta) = -2\sum_{i=1}^{n} (y_i - \theta) = -2\left[\sum_{i=1}^{n} y_i - n\theta\right]$$

$$s_2''(\theta) = 2n$$

$$x^* = x^{(t)} - \frac{g'\left(x^{(t)}\right)}{g''\left(x^{(t)}\right)} = x^{(t)} + h^{(t)}$$

$$h^{(t)} = -\frac{g'\left(x^{(t)}\right)}{g''\left(x^{(t)}\right)} \to -\frac{-2\sum_{i=1}^{n}\left(y_i - \theta^{(t)}\right)}{2n} \boxed{= \frac{\sum_{i=1}^{n}\left(y_i - \theta^{(t)}\right)}{n}}$$

2. Maximize the function $f(x) = -\frac{x^4}{4} + \frac{x^2}{2} - x + 2$ by using Newton's Method and the starting values below. For each value state the number of iterations Newton's Method takes converge if we want our solution to be correct to 2 decimal places.

   The update algorithm is as follows:

$$f'(x) = -x^3 + x - 1$$

$$f''(x) = -3x^2 + 1$$

$$x^{(t+1)} = x^{(t)} + h^{(t)} = x^{(t)} - \frac{-x^3 + x - 1}{-3x^2 + 1}$$

a. $x_0 = -1$

After 4 iterations it converges to approximately -1.32 (code in appendix).

b. $x_0 = 2$

After 64 iterations it converges to approximately -1.32 (code in appendix).

3. Problem 2.1. For part (e) you only need to discuss your results, you do not need to reapply the methods to a random data set.

$$X_1, \cdots, X_n \sim i.i.d. Cauchy(\theta, 1)$$

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

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

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

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

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

Newton-Raphson results:

θ_0	-11	-1	0	1.5	4	4.7	7	8	38
$Iterations$	257	3	2	3	4	4	8	257	4
θ_{final}	-0.1922825	-0.1922865	-0.1922825	1.7135854	2.8174724	-0.1922865	41.0408478	-0.1922825	42.7953779

In the first part when using Newton-Raphson, it's not difficult to utilize a variety of initial starting points. The results of them however can be interesting at times. Even if the initial starting point is visually within the neighborhood of the global maximum on the graph, it's still possible for the algorithm to choose a nearby local maximum instead. This occurred with values such as 1.5 and 4, despite them being quite close to the perceived global maximum. Also, there seemed to be two points that the algorithm would converge to, -1.922825 and -1.922865, with the latter being closer when plotted as a vertical line over the graph of the log-likelihood. This best approximation was found with starting values of -1 and 4.7. In all cases, the algorithm did manage to converge eventually, however in two instances the value it stopped at was quite distant from the rest at 41.04085 and 42.79538. Looking at the graph of the log-likelihood, it makes sense how this could happen since there is a small hill on the graph in that area, indicating a local maximum. In most cases, the speed for convergence was quite fast, happening in less than 10 iterations. There were however two cases where it took 257 iterations. It's difficult to understand exactly why the speed varies, but it did occur for values somewhat further away from any local or global maximum. In other words, they had to do more climbing in order to find the top of a hill.

Bisection Method results:

(a_0, b_0)	$(-1, 1)$	$(-5, 5)$	$(-10, 10)$	$(-20, 40)$
$Iterations$	7	9	10	12
θ_{final}	-0.1953125	-0.1855469	-0.1855469	2.814941

The Bisection Method was one of the simpler approaches. The only requirement is that the objective function is first plotted so that a good estimate of a reasonable interval for the location of the global maximum can be used. Using the first 3 intervals, the algorithm was able to come quite close to where the global maximum is. The last interval of $(-20, 40)$ is relatively wide and so it makes sense why the algorithm would converge to some other local maximum rather than the desired global maximum. Also, a reasonable approach would be to take a rough estimate of the interval and then proceed to choose narrower intervals until the plot of the graph within the region seems to coincide with the approximated location of the global maximum. It's worth noting though that the accuracy of the algorithm is possibly not as good as that of Newton-Raphson, based on the visual appearance of the location of the estimated global maximum relative to the peak of the hill on the graph. It's possible though that with more cases, a more precise interval can be determined, and the same global maximum can be approximated in the end. The speed for these cases were also quite fast, all of them converged within 15 iterations. It makes sense that the larger the interval, the more iterations it would take to converge, a pattern seen above.

Fixed-Point Iterations results:

(α, θ_0)	$(1, -1)$	$(0.64, -1)$	$(0.25, -1)$
$Iterations$	$2{,}065$	114	5
θ_{final}	-0.1977772	-0.1971533	-0.1937823

The Fixed-Point Iterations was one of the more surprising algorithms. In terms of the stability of the algorithm, it was able to converge to a reasonable approximation of the global maximum in all three cases. This is something different from the other algorithms, despite them all being versions of a Fixed-Point Iteration. It's likely however that with other initialization values, there would be approximations that are far less accurate. Also, none of them failed to converge. However, in one of the cases, where $\alpha = 1$, the algorithm took quite a long time before converging. This is an outlier and is longer than any of the other algorithms with an iteration size of 2,065. The speed for when $\alpha = 0.64$ is also a bit longer than average for other algorithms, taking 114 iterations before converging. It's possible that a tradeoff that Fixed-Point Iterations have is that despite the simplicity of the algorithm, it can take much longer to converge.

Secant Method results:

(θ_0, θ_1)	$(-2, -1)$	$(-3, 3)$
$Iterations$	4	4
θ_{final}	-0.1923655	2.817013

The Secant Method is unique compared to the other algorithms in that it requires a past value to update the algorithm. The other algorithms only relied on the current value to continue. In the case with $(-2, -1)$ as the two starting values, the algorithm was able to converge to an accurate position. However, in the other case it converged to some other local maximum. In the former case, the two initial starting values seem to be in a reasonable climbing range, going from -2 to -1. However, in the latter case, the algorithm goes from -3 to 3, skipping over the area where the global maximum appears to be located. This is possibly what made the algorithm fail to converge to a decent approximation in the latter case. The speed however is quite fast for both instances. They both converged at the same time. It's likely that with more distant initializations that the algorithm would take longer to converge.

Comparison:

In the end, it seems like all the algorithms were able to come reasonably close to approximating the global maximum of the log-likelihood of the given Cauchy distribution. It seems however, that with the current amount of work put into tuning the parameters, Newton-Raphson was the closest to correctly approximating the location of the global maximum. The other algorithms perhaps appear weaker due to fewer initial values being tested. However, Newton-Raphson could rely on a set of starting points that can accurately approximate the location to a level that the others perhaps would struggle with. The downside is that it can feel random at times where it's not easy to choose correctly which values to utilize. The Bisection Method felt most intuitive, but it's not clear whether it could approximate to as great of an accuracy as the others. The Fixed-Point Iterations is the simplest in terms of the component of the algorithm but had the most surprising results. It's not clear what the reason is for the slowness, but it's possible that the algorithm functions well but only inefficiently. The Secant Method was a method that worked quickly but didn't always converge to an ideal location. In comparison, it may seem that due to the effort needed to fine tune many of the algorithms, they may be just as difficult to use as Newton-Raphson. Therefore, it may seem that Newton-Raphson is the best due to its perceived accuracy in these cases.

4. In each of the following, assume a random sample of size n, $x_1, x_2, \cdots, x_n$ and find the Maximum Likelihood Estimator for:

   a. λ for the Poisson distribution.

$$X_1, \cdots, X_n \sim i.i.d. Poisson(\lambda);\ x \in \mathbb{N}_0$$

$$L(\lambda) = f(x_1, \cdots, x_n; \lambda) = \prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!} = \frac{\lambda^{\sum_{i=1}^{n} x_i} e^{-n\lambda}}{\prod_{i=1}^{n} x_i!}$$

$$l(\lambda) = \ln L(\lambda) = \sum_{i=1}^{n} x_i \ln \lambda - n\lambda - \ln \prod_{i=1}^{n} x_i!$$

$$l'(\lambda) = \frac{\sum_{i=1}^{n} x_i}{\lambda} - n \overset{set\ to}{\triangleq} 0 \rightarrow \boxed{\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n} = \bar{x}}$$

Then, given that at least one observation is nonzero.

$$l''(\lambda) = \frac{-\sum_{i=1}^{n} x_i}{\lambda^2} \rightarrow \frac{-\sum_{i=1}^{n} x_i}{\bar{x}^2} < 0 \therefore\ \hat{\lambda}\ is\ the\ MLE$$

b. θ for the Exponential distribution.

$$X_1, \cdots, X_n \sim i.i.d. Exponential(\theta);\ x \in [0, \infty)$$

$$L(\theta) = f(x_1, \cdots, x_n; \theta) = \prod_{i=1}^{n} \theta e^{-\theta x_i} = \theta^n e^{-\theta \sum_{i=1}^{n} x_i}$$

$$l(\theta) = \ln L(\theta) = n \ln \theta - \theta \sum_{i=1}^{n} x_i$$

$$l'(\theta) = \frac{n}{\theta} - \sum_{i=1}^{n} x_i \overset{set\ to}{\triangleq} 0 \rightarrow \boxed{\hat{\theta} = \frac{n}{\sum_{i=1}^{n} x_i} = \frac{1}{\bar{x}}}$$

Then, given that at least one observation is nonzero.

$$l''(\theta) = -\frac{n}{\theta^2} \rightarrow -n\bar{x}^2 < 0 \therefore\ \hat{\theta}\ is\ the\ MLE$$

Note: These are examples of distributions for which the MLE can be found analytically in terms of the data $x_1, \cdots, x_n$ and so no advanced computational methods are required.

5. Consider a sequence of n independent Bernoulli trials in which the probability of success is θ and the probability of failure is $1 - \theta$. If A represents the observed number of successes and B represents the observed number of failures, (with $A + B = n$), the find $I(\theta)$, the Fisher information matrix. (Hint: Recall that the sum of n Bernoulli trials is a Binomial random variable. Also assume that n, A and B are fixed and so the only unknown parameter is θ, in the case $I(\theta)$ will be a scalar.)

Let $X_i \sim i.i.d. Bernoulli(\theta)$, then let $X = \sum_{i=1}^{n} X_i \sim i.i.d. Binomial(n, \theta)$.

Then taking the likelihood of X we have:

$$L(\theta) = f(x_1, \cdots, x_n|\theta) = \binom{n}{A}\theta^A(1-\theta)^B = \left(\begin{matrix} n \\ \sum_{i=1}^{n} x_i \end{matrix}\right)\theta^{\sum_{i=1}^n x_i}(1-\theta)^{n-\sum_{i=1}^n x_i}$$

$$l(\theta) = \ln L(\theta) = \ln\left(\begin{matrix} n \\ \sum_{i=1}^{n} x_i \end{matrix}\right) + \left(\sum_{i=1}^{n} x_i\right)\ln\theta + \left(n - \sum_{i=1}^{n} x_i\right)\ln(1-\theta)$$

$$l'(\theta) = \frac{\sum_{i=1}^n x_i}{\theta} - \frac{n - \sum_{i=1}^n x_i}{1-\theta} \overset{set\ to}{\triangleq} 0$$

$$\rightarrow \frac{\sum_{i=1}^n x_i}{\theta} = \frac{n - \sum_{i=1}^n x_i}{1-\theta} \rightarrow \frac{1-\theta}{\theta} = \frac{n - \sum_{i=1}^n x_i}{\sum_{i=1}^n x_i} \rightarrow \frac{1}{\theta} - 1 = \frac{1}{\bar{x}} - 1 \rightarrow \hat{\theta} = \bar{x}$$

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

$$I(\theta) = -E[l''(\theta)] = -E\left[-\frac{\sum_{i=1}^n X_i}{\theta^2} - \frac{n - \sum_{i=1}^n X_i}{(1-\theta)^2}\right]$$

$$= E\left[\frac{\sum_{i=1}^n X_i}{\theta^2}\right] + E\left[\frac{n - \sum_{i=1}^n X_i}{(1-\theta)^2}\right] = \frac{1}{\theta^2}E\left[\sum_{i=1}^{n} X_i\right] + \frac{1}{(1-\theta)^2}\left\{n - E\left[\sum_{i=1}^{n} X_i\right]\right\}$$

$$= \frac{1}{\theta} + \frac{n}{1-\theta} = \frac{n(1-\theta) + n\theta}{\theta(1-\theta)} \boxed{= \frac{n}{\theta(1-\theta)}}$$

Code Appendix:

```r
### Problem 1
library(latex2exp)
y_sample_data <- c(28, 33, 22, 35, 31)  # Sample data
y_sample_mean <- mean(y_sample_data)  # Sample mean
theta_hat <- y_sample_mean  # MLE of theta

L2_norm_squared <- function(y, theta) {  # Objective function
  return(sum((y - theta)^2))
}

theta_vector <- seq(theta_hat - 10, theta_hat + 10, length.out = 200)
L2_values <- sapply(theta_vector, function(index)
  L2_norm_squared(y = y_sample_data, theta = index))
plot(theta_vector, L2_values, type = 'l', main = TeX('$s_p(\\theta)$'),
     ylab = 'L2 norm squared', xlab = TeX('$\\theta$'))

L2_norm_squared_derivative <- function(y, theta) {
  n <- length(y)
  return(-2 * sum(y) + 2 * n * theta)
}

initialize_variables <- function(a_0, b_0) {
  x_0 <- (a_0 + b_0) / 2
  return(c(x_0, a_0, b_0))
}

absolute_convergence_function <- function(x_t, x_update, epsilon) {
  absolute_convergence_criterion <- abs(x_update - x_t)
  bool_flag <- ifelse(absolute_convergence_criterion < epsilon, FALSE, TRUE)
  return(c(absolute_convergence_criterion, bool_flag))
}

a_b_update <- function(a_t, b_t, x_t) {
  if(L2_norm_squared_derivative(y = y, theta = a_t) *
     L2_norm_squared_derivative(y = y, theta = x_t) <= 0) {
    a_update = a_t; b_update = x_t
  } else {
    a_update = x_t; b_update = b_t
  }
  return(c(a_update, b_update))
}

x_update_function <- function(a_update, b_update) {
  x_update <- (a_update + b_update) / 2
  return(x_update)
}

bisection <- function(a, b) {
  initial_vector <- initialize_variables(a_0 = a, b_0 = b)  # Initialize variables
  x_t <- initial_vector[1]; a_t <- initial_vector[2]; b_t <- initial_vector[3]
  absolute_convergence_criterion <- TRUE
  epsilon <- 0.01
  counter <- 0

  while (absolute_convergence_criterion) {  # Loop through algorithm
    a_b_update_vector <- a_b_update(a_t = a_t, b_t = b_t, x_t = x_t)
    a_update <- a_b_update_vector[1]; b_update <- a_b_update_vector[2]

    x_update <- x_update_function(a_update = a_update, b_update = b_update)

    convergence_vector <- absolute_convergence_function(x_t = x_t,
                                    x_update = x_update, epsilon = epsilon)
    absolute_convergence_value <- convergence_vector[1]
    absolute_convergence_criterion <- convergence_vector[2]

    a_t <- a_update; b_t <- b_update; x_t <- x_update; counter <- counter + 1
  }

  cat('iteration t =', counter, '\n')  # Print results
  cat('absolute convergence criterion: ', absolute_convergence_value, '\n')
  cat('x_update: ', x_update, '\n')

  theta <- seq(a, b, length.out = 200)  # Zoom in to check
  w <- sapply(theta, function(z) L2_norm_squared(y = y, theta = z))
  plot(theta, w, main = TeX('$s_p(\\theta)$'), type = 'l')
  legend("topright", legend = c("theta = 30", "bisection update"),
         col = c('blue', 'red'), lty = c(1,1))
  abline(v = 30, col = 'blue')
  abline(v = x_update, col = 'red')
}

bisection(a = 25, b = 35) # 29.79492 (t=9)
bisection(a = 28, b = 32) # 29.80469 (t=8)
bisection(a = 29, b = 30) # 29.80469 (t=6)
bisection(a = 29.6, b = 30) # 29.79375 (t=5)
bisection(a = 29.7, b = 29.9) # 29.79375 (t=4)

### Problem 2
problem_2_first_derivative <- function(x) {
  return(-x^3 + x - 1)
}

problem_2_second_derivative <- function(x) {
  return(-3 * x^2 + 1)
}
```

```r
h <- function(x) {
  return(-problem_2_first_derivative(x) / problem_2_second_derivative(x))
}

nr <- function(x_init, epsilon = 0.01) {
  x_new <- x_init  # Initialize variables
  x_old <- x_init + 9999
  counter <- 0
  absolute_convergence_criterion <- TRUE

  while(absolute_convergence_criterion) {  # Loop through algorithm
    x_old <- x_new
    x_new <- x_old + h(x_old)

    convergence_vector <- absolute_convergence_function(x_t = x_old,
                                              x_update = x_new, epsilon = epsilon)
    absolute_convergence_value <- convergence_vector[1]
    absolute_convergence_criterion <- convergence_vector[2]

    counter <- counter + 1
  }

  cat('iteration t =', counter, '\n')  # Print results
  cat('absolute convergence criterion: ', absolute_convergence_value, '\n')
  cat('x_update: ', x_new, '\n')
}

nr(-1) # -1.324718 (t=4)
nr(2) # -1.324732 (t=64)

problem_2_function <- function(x) {
  return(-(1/4) * x^4 + (1/2) * x^2 - x + 2)
}

xs <- seq(-1.35, -1.3, length.out = 200) # Plot problem 2 and check values
plot(xs, problem_2_function(xs), type = 'l')
abline(v = -1.324718)
abline(v = -1.324732)

### Problem 3
### Newton-Raphson
cauchy_sample_data <- c(1.77, -0.23, 2.76, 3.80, 3.47,
       56.75, -1.34, 4.24, -2.44, 3.29,
       3.71, -2.40, 4.53, -0.07, -1.05,
       -13.87, -2.53, -1.75, 0.27, 43.21)

log_likelihood_cauchy <- function(x, theta) {
  n <- length(x)
  return(-n * log(pi) - sum(log(1 + (x - theta)^2)))
}

theta_sequence <- seq(-50, 50, 0.01)
log_likelihood_values_cauchy <- sapply(theta_sequence, function(index)
  log_likelihood_cauchy(x = cauchy_sample_data, theta = index))
plot(theta_sequence, log_likelihood_values_cauchy,
     type = 'l', main = TeX('$Cauchy log likelihood$'),
     ylab = 'log-likelihood theta', xlab = TeX('$\\theta$'))

log_likelihood_cauchy2 <- function(x, theta) {  # First derivative of log-likelihood
  return(2 * sum((x - theta) / (1 + (x - theta)^2)))
}

log_likelihood_cauchy3 <- function(x, theta) {  # Second derivative of log-likelihood
  squared_difference <- (x - theta)^2
  return(2 * sum(((2 * squared_difference) /
                  (1 + squared_difference)^2) - (1 / (1 + squared_difference))))
}

h_cauchy <- function(x, theta) {  # h(t) function
```

```r
  return(-log_likelihood_cauchy2(x = x, theta = theta) /
           log_likelihood_cauchy3(x = x, theta = theta))
}

nr <- function(x = cauchy_sample_data, theta_init, epsilon = 0.01) {
  theta_new <- theta_init  # Initialize variables
  theta_old <- theta_init + 9999
  counter <- 0
  absolute_convergence_criterion <- TRUE

  while(absolute_convergence_criterion) {  # Loop through algorithm
    theta_old <- theta_new
    theta_new <- theta_old + h_cauchy(x = x, theta = theta_old)

    convergence_vector <- absolute_convergence_function(x_t = theta_old,
                                        x_update = theta_new, epsilon = epsilon)
    absolute_convergence_value <- convergence_vector[1]
    absolute_convergence_criterion <- convergence_vector[2]

    counter <- counter + 1
  }

  cat('theta_init =', theta_init, '\n')
  cat('iteration t =', counter, '\n')  # Print results
  cat('absolute convergence criterion: ', absolute_convergence_value, '\n')
  cat('theta_update: ', theta_new, '\n')
  return(theta_new)
}
starting_points <- c(-11, -1, 0, 1.5, 4,
                     4.7, 7, 8, 38)
theta_approximate <- sapply(starting_points, function(index) nr(theta_init = index))

theta_sequence <- seq(-0.19231, -0.19226, length.out = 200)
log_likelihood_values <- sapply(theta_sequence, function(index)
  log_likelihood_cauchy(x = cauchy_sample_data, theta = index))
plot(theta_sequence, log_likelihood_values,
     type = 'l', main = 'Cauchy log likelihood',
     ylab = 'log-likelihood theta', xlab = TeX('$\\theta$'))
abline(v = theta_approximate[1]) # -0.1922825
abline(v = theta_approximate[2]) # -0.1922865 (closer)
abline(v = theta_approximate[3]) # -0.1922825
abline(v = theta_approximate[6]) # -0.1922865 (closer)
abline(v = theta_approximate[8]) # -0.1922825
# -1, 4.7 closest

### bisection method
a_b_update_cauchy <- function(a_t, b_t, x_t) {
  if(log_likelihood_cauchy2(x = cauchy_sample_data, theta = a_t) *
     log_likelihood_cauchy2(x = cauchy_sample_data, theta = x_t) <= 0) {
    a_update = a_t; b_update = x_t
  } else {
    a_update = x_t; b_update = b_t
  }
  return(c(a_update, b_update))
}

bisection_cauchy <- function(a, b) {
  initial_vector <- initialize_variables(a_0 = a, b_0 = b)  # Initialize variables
  x_t <- initial_vector[1]; a_t <- initial_vector[2]; b_t <- initial_vector[3]
  absolute_convergence_criterion <- TRUE
  epsilon <- 0.01
  counter <- 0

  while (absolute_convergence_criterion) {  # Loop through algorithm
    a_b_update_vector <- a_b_update_cauchy(a_t = a_t, b_t = b_t, x_t = x_t)
    a_update <- a_b_update_vector[1]; b_update <- a_b_update_vector[2]

    x_update <- x_update_function(a_update = a_update, b_update = b_update)
    convergence_vector <- absolute_convergence_function(x_t = x_t,
                                    x_update = x_update, epsilon = epsilon)
    absolute_convergence_value <- convergence_vector[1]
    absolute_convergence_criterion <- convergence_vector[2]

    a_t <- a_update; b_t <- b_update; x_t <- x_update; counter <- counter + 1
  }

  cat('iteration t =', counter, '\n')  # Print results
  cat('absolute convergence criterion: ', absolute_convergence_value, '\n')
  cat('x_update: ', x_update, '\n')

  theta <- seq(a, b, length.out = 200)  # Zoom in to check
  w <- sapply(theta, function(y)
    log_likelihood_cauchy(x = cauchy_sample_data, theta = y))
  plot(theta, w, main = "log likelihood function", type = 'l')
  legend("bottomleft", legend = c("theta = 0", "bisection update"),
         col = c('blue', 'red'), lty = c(1,1))
  abline(v = 0, col = 'blue')
  abline(v = x_update, col = 'red')
}

bisection_cauchy(a = -1, b = 1)  # -0.1953125 (t=7)
bisection_cauchy(a = -5, b = 5)  # -0.1855469 (t=9)
bisection_cauchy(a = -10, b = 10)  # -0.1855469 (t=10)
bisection_cauchy(a = -20, b = 40)  # 2.814941 (t=12)

### fixed-point iterations
fixed_point <- function(theta_init, alpha) {
  theta_current <- theta_init  # Initialize variables
  absolute_convergence_criterion <- TRUE
  epsilon <- 0.01
  counter <- 0

  while (absolute_convergence_criterion) {  # Loop through algorithm
    theta_update <- theta_current + alpha *
      log_likelihood_cauchy2(x = cauchy_sample_data, theta = theta_current)

    convergence_vector <- absolute_convergence_function(x_t = theta_current,
                              x_update = theta_update, epsilon = epsilon)
    absolute_convergence_value <- convergence_vector[1]
    absolute_convergence_criterion <- convergence_vector[2]

    theta_current <- theta_update; counter <- counter + 1
  }

  cat('iteration t =', counter, '\n')  # Print results
  cat('absolute convergence criterion: ', absolute_convergence_value, '\n')
  cat('theta_update: ', theta_update, '\n')

  return(theta_update)
}
alpha_vector <- c(1, 0.64, 0.25)
sapply(alpha_vector, function(index) fixed_point(theta_init = -1, alpha = index))
# -0.1977772 (t=2065, alpha=1)
# -0.1971533 (t=114, alpha=0.64)
# -0.1937823 (t=5, alpha=0.25)

### secant method
secant_update <- function(theta_current, theta_previous) {
  theta_update <- theta_current -
    log_likelihood_cauchy2(cauchy_sample_data, theta_current) *
    ((theta_current - theta_previous) /
       (log_likelihood_cauchy2(cauchy_sample_data, theta_current) -
          log_likelihood_cauchy2(cauchy_sample_data, theta_previous)))
  return(theta_update)
}
secant_method <- function(theta_init, theta_1) {
  theta_previous <- theta_init  # Initialize variables
  theta_current <- theta_1
  absolute_convergence_criterion <- TRUE
  epsilon <- 0.01
  counter <- 0

  while (absolute_convergence_criterion) {  # Loop through algorithm
    theta_update <- secant_update(theta_current, theta_previous)

    convergence_vector <- absolute_convergence_function(x_t = theta_current,
                                          x_update = theta_update, epsilon = epsilon)
    absolute_convergence_value <- convergence_vector[1]
    absolute_convergence_criterion <- convergence_vector[2]

    theta_previous <- theta_current; theta_current <- theta_update; counter <- counter + 1
  }

  cat('iteration t =', counter, '\n')  # Print results
  cat('absolute convergence criterion: ', absolute_convergence_value, '\n')
  cat('theta_update: ', theta_update, '\n')

  return(theta_update)
}

secant_method(theta_init = -2, theta_1 = -1) # -0.1923655 (t=4)
secant_method(theta_init = -3, theta_1 = 3) # 2.817013 (t=4)
```