Principles of statistical inference project - Part 1

Carmel Gafa'

1 Question 1

Consider a random variable X that follows an exponential distribution with scale parameter λ.

The **Exponential Distribution** is a continuous probability distribution that represents the time intervals between consecutive events in a Poisson process, where events happen independently and at a constant average rate. It is defined by a single parameter, λ, referred to as the rate parameter.

1.1 Give reference to a publication in which the exponential distribution has been used in practise. Explain the context in which this distribution has been used in this publication.

Mahmud et al. presented a study where they analyzed and estimated response times to questions on Twitter [1]. The authors developed predictive models to estimate response wait times, exploring three different approaches:

- Personalized wait time models: These models estimate the wait time for a specific user based on their individual history of response wait times. They assume each response event for a user occurs continuously and independently at a constant average rate, modelled by an exponential distribution. Each user's rate parameter λ is estimated as the inverse of their average past response wait times.

  These models demonstrated a promising ability to estimate response times on Twitter. They generally outperformed generalized models and showed reasonable accuracy, especially for an hour or more time limits. The choice of cut-off probability (a threshold used to determine whether a user is considered sufficiently likely to respond to a question on Twitter within a given period) significantly influenced the precision and recall of the predictions.

- Generalized wait time models: Instead of individual models, a single model is built using the previous responses of all users in the dataset, again using the exponential distribution. The rate parameter λ is estimated from the responses of all users. This model underperformed compared to the personalized models in estimating response times on Twitter.

- Time-sensitive wait time models: These models incorporate sensitivity to the time of day or day of the week when questions are sent for both generalized and personalized models by calculating the rate parameter based on responses to questions sent during a specific day or hour. Personalized time-sensitive models only considered users with at least five responses during the modelled time interval. Incorporating time sensitivity had a modest positive impact on the generalized models but did not consistently improve the performance of the personalized models.

1.2 State the mean and the variance of X.

As X follows an exponential distribution with scale parameter, $X \sim Exp(X)$, the expected value or mean is:

$$E[X] = \frac{1}{\lambda} \tag{1}$$

and the variance is:

$$Var(X) = \frac{1}{\lambda^2} \tag{2}$$

1.3 Derive the moment estimator of λ.

The p.d.f. for an exponential distribution is:

$$f(x) = \lambda e^{-\lambda x}, x \leq 0, \lambda > 0 \tag{3}$$

The first moment, or expected value:

$$\begin{aligned} E[X] &= \int_0^\infty x f(x) dx \\ &= \int_0^\infty x \lambda e^{-\lambda x} dx \\ &= \lambda \int_0^\infty x e^{-\lambda x} \end{aligned}$$

Integrating by parts, we let:
$u = x$ $du = (1)dx$
$v = -\frac{e^{-\lambda x}}{\lambda}$ $dv = e^{-\lambda x} dx$

As $\int u dv = uv - \int v du$:

$$
\begin{aligned}
E[X] &= \lambda \left(\left[-\frac{xe^{-\lambda x}}{\lambda} \right]_0^\infty - \int_0^\infty -\frac{e^{-\lambda x}}{\lambda}(1)dx \right) \\
&= \left[-xe^{-\lambda x} \right]_0^\infty + \int_0^\infty -e^{-\lambda x} dx
\end{aligned}
$$

Let us consider $\left[xe^{-\lambda x} \right]_0^\infty$:

- $-xe^{-\lambda x} = 0$, when $x = 0$
- $\lim\limits_{x \to \infty} xe^{-\lambda x} = 0$, as exponential decay dominates polynomial growth

So the first term is removed;

$$
\begin{aligned}
E[X] &= \int_0^\infty -e^{-\lambda x} dx \\
&= \left[-\frac{1}{\lambda} - e^{-\lambda x} \right]_0^\infty \\
&= 0 - \left(-\frac{1}{\lambda} \right) \\
&= \frac{1}{\lambda}
\end{aligned}
$$

As in the method of moments the sample mean is equal to theoretical expectation;

$$
E[X] = \frac{1}{\lambda} = \overline{X}
$$

and solving for λ

$$
\hat{\lambda} = \frac{1}{\overline{X}} \tag{4}
$$

1.4 Use the second moment to obtain another estimator of λ

For an exponential distribution with rate parameter λ, the second moment,

$$
\begin{aligned}
E[X^2] &= \int_0^\infty x^2 f(x) dx \\
&= \int_0^\infty x^2 \lambda e^{-\lambda x} dx \\
&= \lambda \int_0^\infty x^2 e^{-\lambda x}
\end{aligned}
$$

We let:

$u = x^2 \qquad du = 2xdx$

$v = -\frac{e^{-\lambda x}}{\lambda} \qquad dv = e^{-\lambda x}dx$

$$\begin{aligned} E[X^2] &= \lambda \left(\left[-\frac{x^2 e^{-\lambda x}}{\lambda} \right]_0^\infty - \int_0^\infty -\frac{e^{-\lambda x}}{\lambda} 2xdx \right) \\ &= \left[-x^2 e^{-\lambda x} \right]_0^\infty + \int_0^\infty 2xe^{-\lambda x} dx \end{aligned}$$

Let us consider $\left[x^2 e^{-\lambda x}\right]_0^\infty$:

- $-x^2 e^{-\lambda x} = 0$, when $x = 0$
- $\lim_{x \to \infty} x^2 e^{-\lambda x} = 0$, as exponential decay dominates polynomial growth

Then

$$E[X^2] = \int_0^\infty 2xe^{-\lambda x} dx$$

We let:

$u = x \qquad du = dx$

$v = -\frac{e^{-\lambda x}}{\lambda} \qquad dv = e^{-\lambda x}dx$

$$E[X^2] = 2 \left(\left[-\frac{xe^{-\lambda x}}{\lambda} \right] - \int_0^\infty e^{-\lambda x} dx \right)$$

We have seen previously that the first term will equate to zero.

$$\begin{aligned} E[X^2] &= 2 \int_0^\infty \frac{e^{-\lambda x}}{\lambda} dx \\ &= \left[-\frac{2e^{-\lambda x}}{\lambda^2} \right]_0^\infty \\ &= 0 - \left(-\frac{2}{\lambda^2} \right) \\ &= \frac{2}{\lambda^2} \end{aligned}$$

The variance of the exponential distribution is given by:

$$\begin{aligned} Var(X) &= E[X^2] - E[X]^2 \\ &= \frac{2}{\lambda^2} - \left(\frac{1}{\lambda} \right)^2 \\ &= \frac{1}{\lambda^2} \end{aligned}$$

We can estimate the sample variance

$$\hat{\sigma}^2 = \frac{1}{\hat{\lambda}^2}$$

and solving for λ

$$\hat{\lambda} = \frac{1}{\sqrt{\hat{\sigma}^2}} \tag{5}$$

1.5 Comment on the unbiasedness and consistency of the moment estimator for λ derived in Q1iii. State any assumption/s that need to be made to check for unbiasedness and consistency.

The moment estimator $\hat{\lambda}$ is both unbiased and consistent.

Unbiasedness: The estimator $\hat{\lambda}$ is unbiased if its expectation equals the true parameter λ:

$$E[\hat{\lambda}] = E\left[\frac{1}{\bar{X}}\right] = \lambda. \tag{6}$$

Since the expectation of the sample mean $\bar{X}$ for an exponential distribution satisfies $E[\bar{X}] = \frac{1}{\lambda}$, applying Jensen's inequality confirms that the moment estimator is unbiased.

Consistency: The estimator $\hat{\lambda}$ is consistent if its variance decreases to zero as $n \to \infty$. The variance of $\hat{\lambda}$ is given by:

$$\mathrm{Var}(\hat{\lambda}) = \frac{\lambda^2}{n}. \tag{7}$$

Since $\frac{\lambda^2}{n} \to 0$ as $n \to \infty$, it follows that $\hat{\lambda}$ is a consistent estimator of λ.

1.6 Use R software to generate 1000 data points from an exponential distributed random variable using any admissible parameter value for λ

The R script generates 1000 points from an exponentially distributed random variable with a rate parameter λ of 1.5. It then plots a histogram of these points and overlays the theoretical density function of the exponential distribution. The result is shown in Figure 1.

```
library(ggplot2)
library(glue)

set.seed(50)
lambda <- 1.5
x <- rexp(n = 1000, rate = lambda)

data <- data.frame(x = x)

p <- ggplot(data,
                aes(x = x)) +
        geom_histogram(
                aes(y = after_stat(density)),
                bins = 50, fill = "blue",
                color = "black",
                alpha = 0.6) +
        stat_function(
                fun = function(x) lambda * exp(-lambda * x),
                color = "red",
                size = 1) +
        labs(title = glue("Histogram of exponentially distributed
        random variable with lambda = {lambda}"),
                x = "x", y = "Density") +
        theme_minimal() +
        theme(plot.title = element_text(hjust = 0.5))
```

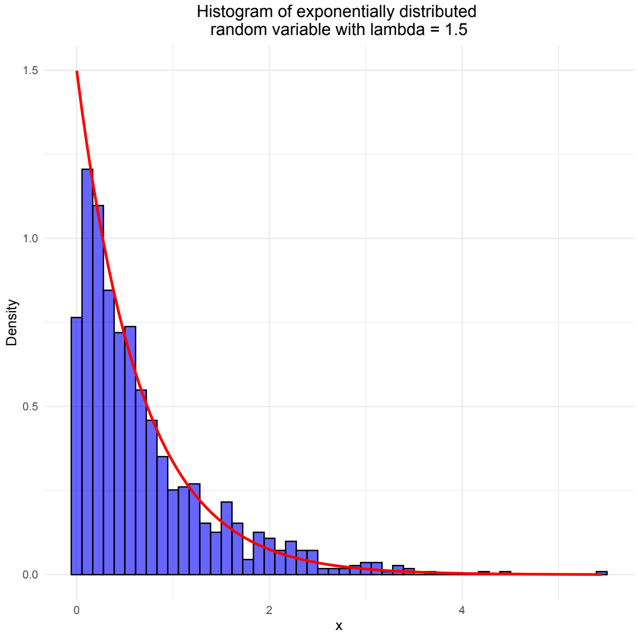


Figure 1: Exponential distributed random variable; histogram of 1000 generated points and theoretical distribution

1.6.1 Write down the log-likelihood function for this exponentially distributed random variable.

For sample $\mathbf{x} = (x_1, \dots, x_n)^T$ obtained on an exponential distributed random variable X, with parameter vector $\theta = (\lambda)$, the likelihood

$$\begin{aligned} L(\mathbf{x}, \theta) &= \prod_{i=1}^{n} f(x_i, \theta) \\ &= \prod_{i=1}^{n} \lambda e^{-\lambda x_i} \\ &= \lambda^n e^{\sum_{i=1}^{n} \lambda x_i} \end{aligned}$$

The log-likelihood is then

$$l(\mathbf{x}, \theta) = n \log(\lambda) - \lambda \sum_{i=0}^{n} x_i$$

Taking the derivative with respect to λ,

$$\frac{\partial l(\mathbf{x}, \theta)}{\partial \lambda} = \frac{n}{\lambda} - \sum_{i=0}^{n} x_i$$

for maximum $\frac{\partial l(\mathbf{x}, \theta)}{\partial \lambda}$

$$\begin{aligned} L\frac{\partial l(\mathbf{x}, \theta)}{\partial \lambda} &= 0 \\ \frac{n}{\lambda} - \sum_{i=0}^{n} x_i &= 0 \\ \frac{n}{\lambda} &= \sum_{i=0}^{n} x_i \end{aligned}$$

So that

$$\hat{\lambda} = \frac{n}{\sum_{i=0}^{n} x_i} = \frac{1}{\overline{x}} \tag{8}$$

1.6.2 Evaluate the log-likelihood function for the generated data as a function of λ, and plot the resulting log-likelihood function against different values of λ. Present the plot together with the answers.

The following listing calculates and plots the log-likelihood values for an exponential distribution with varying λ values. The resulting plot can be examined in Figure 2

```
lambda_values <- seq(0.1, 5, by = 0.01)
log_likelihood_values <- sapply(lambda_values,
function(lambda) LL_exponential(lambda, x))

df <- data.frame(lambda_values, log_likelihood_values)

p <- ggplot(df,
        aes(x = lambda_values,
        y = log_likelihood_values)) +
geom_point(
        color = "blue",
        alpha = 0.6) +
        labs(
        title = "Log-Likelihood with varying
 lambda values",
        x = "Lambda",
        y = "Log-Likelihood") +
theme_bw()

quartz()
print(p)
```

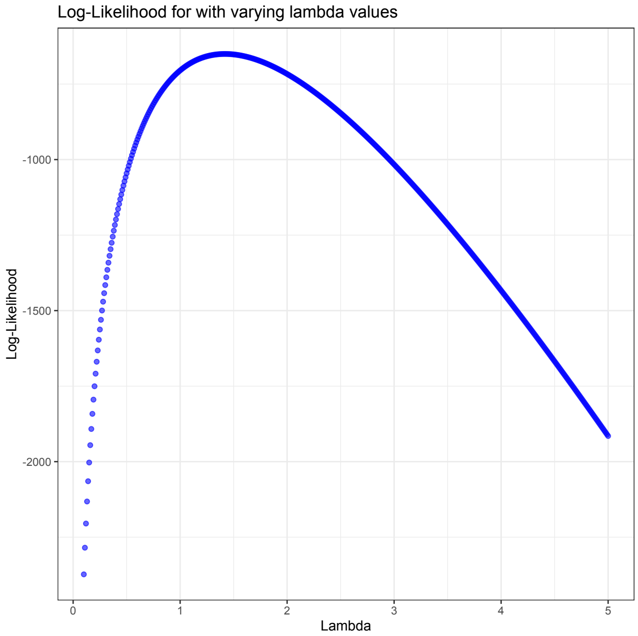


Figure 2: Log-likelihood plot for an exponential distribution with varying λ

1.6.3 Using the plot or otherwise, which estimate for λ is the MLE? Give a reason for your answer.

As we have seen in the previous question, the maximum likelihood estimation is the value for which the derivative of the log-likelihood with respect to lambda is zero, that is the peak of the curve shown in Figure 2. This can be easily calculated using the code below. The value obtained for $\hat{\lambda}$ was **1.43**.

```r
max_ll <- max(log_likelihood_values)
max_lambda <- lambda_values[log_likelihood_values == max_ll]
print(glue("Lambda value for maximum log-likelihood is
    {max_lambda}"))
```

2 Question 2

Suppose that we wish to estimate the parameters of the model:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3}^3 + \varepsilon_i$$

given a sample of size n.

2.1 State the type of estimator that needs to be used in this situation and mention two properties of this estimator.

The appropriate estimator for the given regression model:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3}^3 + \varepsilon_i$$

is the Ordinary Least Squares (OLS) Estimator. This estimator is used because it is linear in parameters, even though x_3^3introduces a nonlinear transformation. In addition, OLS minimizes the sum of squared residuals to obtain the best fit for the parameters. Properties of the OLS estimator include:

- Unbiasedness. The OLS estimator is unbiased, meaning its expected value equals the true parameter value; $E[\hat{\beta}] = \beta$.
- Best Linear Unbiased Estimator. OLS provides the lowest variance among all linear unbiased estimators, making OLS the most efficient estimator when errors have constant variance (homoscedasticity).

2.2 Derive the equations that need to be solved to obtain the required estimates.

For the given model:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3}^3 + \varepsilon_i$$

The sum os squared residuals will be minimized using ordinary least squares.

$$S = \sum_{i=1}^{n} (\varepsilon_i = \sum_{i-1}^{n} y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3}^3)$$

Our aim is to find

$$\arg \max_{\beta_0,\beta_1,\beta_2,\beta_3} S$$

by equating

$$\frac{\partial S}{\partial \beta_0} = 0, \frac{\partial S}{\partial \beta_1} = 0, \frac{\partial S}{\partial \beta_2} = 0, \frac{\partial S}{\partial \beta_3} = 0$$

that is,

$$\begin{aligned}
\frac{\partial S}{\partial \beta_0} =& 2\sum_{i=1}^{n} (\varepsilon_i = \sum_{i-1}^{n} y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3}^3) = 0 \\
\frac{\partial S}{\partial \beta_1} =& \sum_{i=1}^{n} (\varepsilon_i = \sum_{i-1}^{n} y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3}^3)(-x_{i1}) = 0 \\
\frac{\partial S}{\partial \beta_2} =& \sum_{i=1}^{n} (\varepsilon_i = \sum_{i-1}^{n} y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3}^3)(-x_{i2}) = 0 \\
\frac{\partial S}{\partial \beta_3} =& \sum_{i=1}^{n} (\varepsilon_i = \sum_{i-1}^{n} y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3}^3)(-x_{i3}^3) = 0
\end{aligned}$$

rearranging,

$$\begin{aligned}
\sum_{i=1}^{n} y_i =& n\beta_0 + \beta_1 \sum_{i=1}^{n} x_{i1} + \beta_2 \sum_{i=1}^{n} x_{i2} + \beta_3 \sum_{i=1}^{n} x_{i_3}^3 \\
\sum_{i=1}^{n} y_i x_{i1} =& \beta_0 \sum_{i=1}^{n} x_{i1} + \beta_1 \sum_{i=1}^{n} x_{i1}^2 + \beta_2 \sum_{i=1}^{n} x_{i2} x_{i1} + \beta_3 \sum_{i=1}^{n} x_{i_3}^3 x_{i1} \\
\sum_{i=1}^{n} y_i x_{i2} =& \beta_0 \sum_{i=1}^{n} x_{i2} + \beta_1 \sum_{i=1}^{n} x_{i1} x_{i2} + \beta_2 \sum_{i=1}^{n} x_{i2}^2 + \beta_3 \sum_{i=1}^{n} x_{i_3}^3 x_{i2} \\
\sum_{i=1}^{n} y_i x_{i_3}^3 =& \beta_0 \sum_{i=1}^{n} x_{i_3}^3 + \beta_1 \sum_{i=1}^{n} x_{i1} x_{i_3}^3 + \beta_2 \sum_{i=1}^{n} x_{i2} x_{i_3}^3 + \beta_3 \sum_{i=1}^{n} x_{i_3}^6
\end{aligned}$$

that can be written in matrix form as follows:

$$\begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} y_i x_{i1} \\ \sum_{i=1}^{n} y_i x_{i2} \\ \sum_{i=1}^{n} y_i x_{i_3}^3 \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^{n} x_{i1} & \sum_{i=1}^{n} x_{i2} & \sum_{i=1}^{n} x_{i_3}^3 \\ \sum_{i=1}^{n} x_{i1} & \sum_{i=1}^{n} x_{i1}^2 & \sum_{i=1}^{n} x_{i2} x_{i1} & \sum_{i=1}^{n} x_{i_3}^3 x_{i1} \\ \sum_{i=1}^{n} x_{i2} & \sum_{i=1}^{n} x_{i1} x_{i2} & \sum_{i=1}^{n} x_{i2}^2 & \sum_{i=1}^{n} x_{i_3}^3 x_{i2} \\ \sum_{i=1}^{n} x_{i_3}^3 & \sum_{i=1}^{n} x_{i1} x_{i_3}^3 & \sum_{i=1}^{n} x_{i2} x_{i_3}^3 & \sum_{i=1}^{n} x_{i_3}^6 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \tag{9}$$

Equation 9 needs to be solved to obtain the required estimates. But let us try to move this a bit further. If we ser the design matrix $\mathbf{X}$ as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ 1 & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n3} \end{bmatrix}$$

We can see that

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} n & \sum_{i=1}^n x_{i1} & \sum_{i=1}^n x_{i2} & \sum_{i=1}^n x_{i_3}^3 \\ \sum_{i=1}^n x_{i1} & \sum_{i=1}^n x_{i1}^2 & \sum_{i=1}^n x_{i2}x_{i1} & \sum_{i=1}^n x_{i_3}^3 x_{i1} \\ \sum_{i=1}^n x_{i2} & \sum_{i=1}^n x_{i1}x_{i2} & \sum_{i=1}^n x_{i2}^2 & \sum_{i=1}^n x_{i_3}^3 x_{i2} \\ \sum_{i=1}^n x_{i_3}^3 & \sum_{i=1}^n x_{i1}x_{i_3}^3 & \sum_{i=1}^n x_{i2}x_{i_3}^3 & \sum_{i=1}^n x_{i_3}^6 \end{bmatrix}$$

and

$$\mathbf{X}^T\mathbf{Y} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n y_i x_{i1} \\ \sum_{i=1}^n y_i x_{i2} \\ \sum_{i=1}^n y_i x_{i_3}^3 \end{bmatrix}$$

Therefore Equation 9 can be written as

$$\mathbf{X}^T\mathbf{Y} = \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \tag{10}$$

so that

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{11}$$

$\mathbf{X}^T\mathbf{X}$ is known as the Gram matrix and represents the relationships among predictor variables in a regression model. Its inverse, $(\mathbf{X}^T\mathbf{X})^{-1}$, plays a crucial role in calculating the variance and standard errors of the estimated regression coefficients. Specifically, the diagonal elements of $(\mathbf{X}^T\mathbf{X})^{-1}$ are used to compute the standard errors of the estimated coefficients. The standard error of $\hat{\beta}_j$ is given by:

$$SE(\hat{\beta}_j) = \sqrt{\sigma^2 \cdot ((\mathbf{X}^T\mathbf{X})^{-1})_{jj}}$$

where σ^2 is the residual variance, and $\left((\mathbf{X}^T\mathbf{X})^{-1}\right)_{jj}$ is the diagonal element corresponding to β_j. This formulation ensures that the uncertainty in the coefficient estimates accounts for correlations between predictors, making it a fundamental component in statistical inference for regression models.

2.3 Generate or source a dataset for which you would use this type of model and fit the model to the data. Present a plot of the fitted model to the data. Comment on the goodness of fit of the model to the data. The R code used for this question should be presented together with the answers. Proper referencing should be provided in the text if the data is sourced.

Listing 1 creates a synthetic dataset for this examples and fits the specified linear model. It can be summarized as follows:

1. The predictor variable ranges are first defined. The predictor variables are then generated with random noise to introduce variability, which ensures numerical stability in regression computations.

2. The regression coefficients are then specified; $\beta_0 = 5$, $\beta_1 = 2.5$, $\beta_2 = -1.2$, $\beta_3 = 0.8$

3. The response variable (y) is computed as a linear combination of x_1 and x_2, along with a cubic transformation of x_3 to capture nonlinear effects.

4. Random noise is added to the response variable to simulate measurement error and external variability, forcing the model to account for stochastic influences.

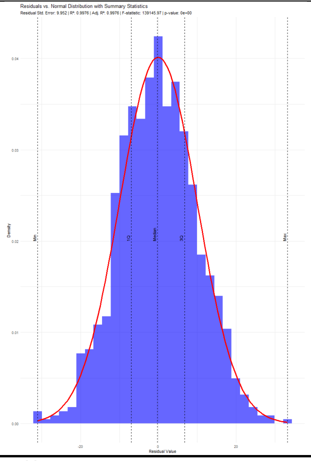
5. A multiple regression model is then fitted to estimate the relationships between predictors and the response variable while incorporating the cubic transformation of x_3 to capture nonlinear patterns.

```
set.seed(2023)

library(ggplot2)

# Generate data

number_of_samples <- 1000

limits_x1 <- c(-10, 10)
limits_x2 <- c(-7, 5)
limits_x3 <- c(-3, 8)

x1 <- seq(limits_x1[1], limits_x1[2], length.out = number_of_samples) + rnorm(number_of_samples, mean = 0, sd = 2)  # Add slight randomness
x2 <- seq(limits_x2[1], limits_x2[2], length.out = number_of_samples) + rnorm(number_of_samples, mean = 0, sd = 2)  # Add slight randomness
x3 <- seq(limits_x3[1], limits_x3[2], length.out = number_of_samples) + rnorm(number_of_samples, mean = 0, sd = 2)  # Add slight randomness

beta_0  <- 5
beta_1  <- 2.5
beta_2  <- -1.2
beta_3  <- 0.8

y <- beta_0 + (beta_1 * x1) + (beta_2 * x2) + (beta_3 * x3^3)
error <- rnorm(length(y), mean = 0.1, sd = 10)  # Random noise
y_real <- y + error


model <- lm(y_real ~ x1 + x2 + I(x3^3))
print(summary(model))
```

Listing 1: Data generation and model fitting using OLS

The following table lists the `summary` results that were obtained. Figure 3 shows the actual vs. predicted values for our model. The plot suggests that the model's predictions are very close to the actual values, meaning it captures most of the variance in the data.

Regression Output	**Comment**
Call	
`lm(formula = y_real   x1 + x2 + I(x3^3))`	Specifies a linear regression where y is predicted using x_1, x_2, and x_3^3. The function `I(x3^3)` ensures x_3 is cubed as a transformation.
Residuals	
`Min = -31.031`	Smallest residual (largest over-prediction).
`1Q = -6.926`	First quartile (25% of residuals are below this).

`Median = -0.218`	Median residual, close to zero suggests unbiased predictions.
`3Q = 6.738`	Third quartile (75% of residuals are below this).
`Max = 33.252`	Largest residual (largest under-prediction).
`residuals plot` (not part of summary but generated subsequently)	
Coefficients	
`(Intercept) = 5.44611`	$\hat{\beta}_0$
`x1 = 2.67591`	$\hat{\beta}_1$
`x2 = -1.41965`	$\hat{\beta}_2$
`I(x3^3) = 0.80047`	$\hat{\beta}_3$
Standard Errors (Precision of Estimates)	
`Intercept SE` = 0.39241	Standard error for β_0. Smaller SE suggests a reliable estimate.
`x1 SE` = 0.09603	Standard error for β_1. A small SE implies that estimates are stable.
`x2 SE` = 0.13976	Standard error for β_2. Suggests moderate precision but check for multicollinearity.
`I(x3^3) SE` = 0.00157	Very small SE, indicating high precision and a narrow confidence interval.
t-values (Test Statistic for Significance)	
`Intercept t-value` = 13.88	Indicates the intercept is statistically significant.
`x1 t-value` = 27.86	Strong evidence that x_1 affects y. Practical significance should also be considered.
`x2 t-value` = -10.16	Strong evidence that x_2 affects y. Multicollinearity should be assessed.
`I(x3^3) t-value` = 509.81	Extremely strong effect of x_3^3 on y. Consider checking residual plots for outliers.
p-values (Significance Levels)	
`Intercept p-value < 2e-16`	Highly significant (`p < 0.001`).
`x1 p-value < 2e-16`	Highly significant (`p < 0.001`).

`x2 p-value < 2e-16`	Highly significant (`p < 0.001`).
`I(x3^3) p-value < 2e-16`	Highly significant (`p < 0.001`).
Model Fit Statistics	
`Residual standard error = 9.952`	Measures average prediction error.
`Multiple` $R^2 = 0.9976$	Model explains 99.76% of the variance in y.
`Adjusted` $R^2 = 0.9976$	Adjusted for the number of predictors, still very high.
`F-statistic =` 1.391×10^5	Tests overall model significance.
`Model p-value < 2.2e-16`	Overall model is highly significant.

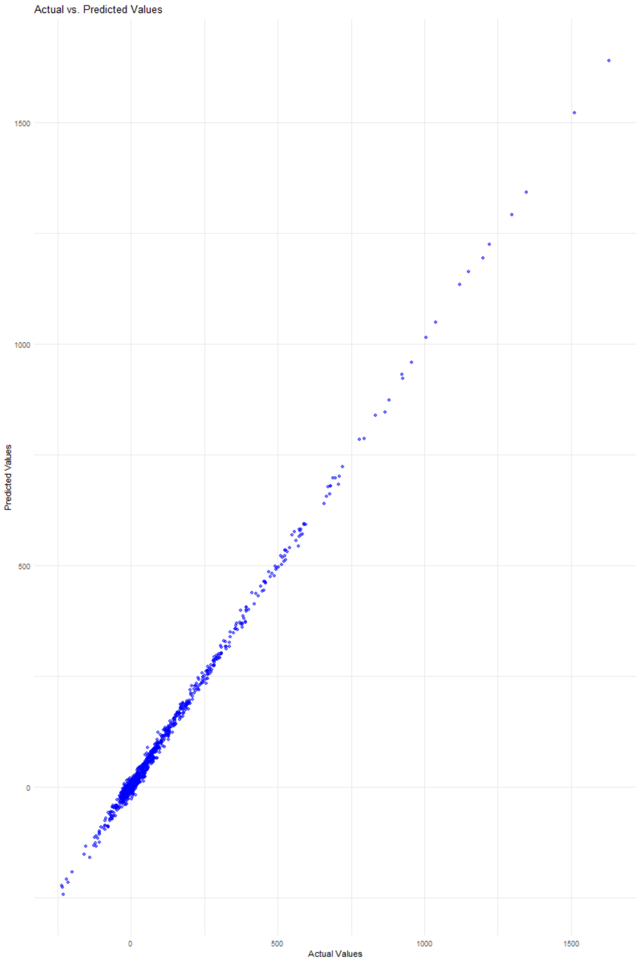


Figure 3: Predicted vs. actual values of the response variable

3 Question 3

Let $x_1, \ldots, x_n$ be a random sample selected from a population with a distribution of your choice. [This distribution needs to be different from those used in the lecture notes].

3.1 Derive the maximum likelihood estimator/s of the parameters of the chosen distribution.

For this problem we will examine the famous Pareto distribution that is used to describe a wide range of phenomena. The probability density function for this distribution is given by;

$$f(x;\alpha,x_m) = \begin{cases} \frac{\alpha x_m^\alpha}{x^{\alpha+1}} & x \geq x_m \\ 0 & otherwise \end{cases} \quad \text{for } \alpha > 0 \quad x_m > 0,$$

Where

- The shape parameter or tail index α defines thickness of the distribution tail.
- The scale parameter x_m represents the minimum possible value that random variable X can take in the distribution.

The likelihood function for this distribution;

$$L(\alpha, x_m; x) = \prod_{i=1}^{n} f(x;\alpha,x_m) = \prod_{i=1}^{n} \frac{\alpha x_m^\alpha}{x_i^{\alpha+1}}$$

then,

$$\begin{aligned} l(\alpha, x_m; x) &= \sum_{i=1}^{n} \log\left(\frac{\alpha x_m^\alpha}{x_i^{\alpha+1}}\right) \\ &= \sum_{i=1}^{n} \left(\log(\alpha x_m^\alpha) - \log(x_i^{\alpha+1})\right) \\ &= \sum_{i=1}^{n} \log(\alpha x_m^\alpha) - \sum_{i=1}^{n} \log(x_i^{\alpha+1}) \\ &= \sum_{i=1}^{n} \log(\alpha) + \sum_{i=1}^{n} \log(x_m^\alpha) - \sum_{i=1}^{n} \log(x_i^{\alpha+1}) \\ &= n\log(\alpha) + n\alpha\log(x_m) - (\alpha+1)\sum_{i=1}^{n} \log(x_i) \end{aligned}$$

As the probability density function is only defined for $x_i \geq x_m$, the likelihood is zero iff $x_m \leq \min(x_1 \dots x_n)$. So, the maximum likelihood estimate of x_m is the largest value such that all observed data are $\geq x_m$. This implies:

$$\widehat{x}_m = \min(x_1 \dots x_n)$$

Similarly,

$$\frac{\partial l(\alpha, x_m; x)}{\partial \alpha} = \frac{n}{\alpha} + n\ \log(x_m) - \sum_{i=1}^{n} \log(x_i)$$

for maximum likelihood

$$\begin{aligned}
\frac{n}{\alpha} + n\ \log(x_m) - \sum_{i=1}^{n} \log(x_i) &= 0 \\
-\frac{n}{\alpha} - n\ \log(x_m) + \sum_{i=1}^{n} \log(x_i) &= 0 \\
-\frac{n}{\alpha} - \sum_{i=1}^{n} \log(x_m) + \sum_{i=1}^{n} \log(x_i) &= 0 \\
-\frac{n}{\alpha} + \sum_{i=1}^{n} \log\frac{x_i}{x_m} &= 0
\end{aligned}$$

Hence;

$$\widehat{\alpha} = \frac{n}{\sum_{i=1}^{n} \log \frac{x_i}{x_m}}$$

Where $x_i \geq x_m$ and $\widehat{x}_m = \min(\mathbf{x})$, so,

$$\widehat{\alpha} = \frac{n}{\sum_{i=1}^{n} \log \frac{x_i}{\min(x_1 \dots x_n)}}$$

Hence the maximum likelihood estimators are:

$$\widehat{x}_m = \min(x_1 \dots x_n), \quad \widehat{\alpha} = \frac{n}{\sum_{i=1}^{n} \log \frac{x_i}{\min(x_1 \dots x_n)}}$$

3.2 Find the Cramer-Rao lower bound for the maximum likelihood estimator/s obtained in Q3i).

The likelihood for a single observation

$$L(x; \alpha, x_m) = \frac{\alpha x_m^{\alpha}}{x^{\alpha+1}}; \quad x \geq x_m$$

The log likelihood

$$l(x; \alpha, x_m) = \log(\alpha) + \alpha \log(x_m) - (\alpha + 1) log(x)$$

For x_m, the score function

$$\frac{\partial l(x;\alpha,x_m)}{\partial x_m} = \frac{\alpha}{x_m}; \quad x \geq x_m, x_m > 0, \alpha > 0$$

Taking the second derivative with respect to x_m

$$\frac{\partial^2 l(x;\alpha,x_m)}{\partial x_m^2} = -\frac{\alpha}{x_m^2}; \quad x \geq x_m, x_m > 0, \alpha > 0$$

The expected value;

$$E\left[\frac{\partial^2 l(x;\alpha,x_m)}{\partial x_m^2}\right] = -\frac{\alpha}{x_m^2}; \quad x \geq x_m, x_m > 0, \alpha > 0$$

Fisher information for n samples,

$$I_n(x_m) = \frac{n\alpha}{x_m^2}; \quad x \geq x_m, x_m > 0, \alpha > 0$$

So CRLB,

$$Var(\widehat{x}_m) \geq \frac{x_m^2}{n\alpha}; \quad x_m > 0, \alpha > 0$$

For α, the score function

$$\frac{\partial l(x;\alpha,x_m)}{\partial \alpha} = \frac{1}{\alpha} + log(x_m) - log(x); \quad x \geq x_m, x_m > 0, \alpha > 0$$

Taking the second derivative with respect to α

$$\frac{\partial^2 l(x;\alpha,x_m)}{\partial \alpha^2} = -\frac{1}{\alpha^2}; \quad x \geq x_m, x_m > 0, \alpha > 0$$

The expected value;

$$E\left[\frac{\partial^2 l(x;\alpha,x_m)}{\partial \alpha^2}\right] = -\frac{1}{\alpha^2}; \quad x \geq x_m, x_m > 0, \alpha > 0$$

Fisher information for n samples,

$$I_n(\alpha) = \frac{n}{\alpha^2}; \quad x \geq x_m, x_m > 0, \alpha > 0$$

So CRLB,

$$Var(\widehat{\alpha}) \geq \frac{\alpha^2}{n}; \quad \alpha > 0$$

3.3 Does/do the ML estimator/s obtained in Q3i) attain the Cramer-Rao lower bound? Give a reason for your answer.

3.4 Is/are the ML estimator/s obtained in Q3i) a sufficient statistic for the population parameter/s? Give a reason for your answer.

3.5 Why are maximum likelihood estimators considered to be desirable estimators? Mention one situation where a maximum likelihood estimator might not be optimal.

4 Question 4 - Jackknife and bootstrap

Consider 50 observations of bivariate pair (X, Y) in resampling.xlsx. Use the nls command in R to estimate the nonlinear regression $Y = \frac{aX}{b+X} + \epsilon$.

The code in Listing 2 performs non-linear regression on the dataset. The resulting plots are presented in Figure 4. The estimated parameters are $\hat{a} = 14.56$ and $\hat{b} = 7.10$.

```
library(openxlsx)
library(ggplot2)

# load file
script_dir <- getwd()
file_path <- file.path(script_dir , "resampling.xlsx")
df <- read.xlsx(file_path , colNames = TRUE)

print(c("number of rows: ", nrow(df)))

# estimate the parameters of the model
init_a <- 1
init_b <- 1

nls_model <- nls(y ~ (a * x) / (b + x),
data = df,
start = list(a = init_a , b = init_b))


estimated_params <- coef(nls_model)
a_hat <- estimated_params["a"]
b_hat <- estimated_params["b"]
cat("Estimated a:", a_hat , "\n")
cat("Estimated b:", b_hat , "\n")

# predict the values of y
df$Predicted <- predict(nls_model)
print(head(df))

# plot the data
p <- ggplot(df, aes(x = x   , y = y)) +
geom_point(color = "blue", alpha = 0.5) +
geom_line(aes(
        y = Predicted),
        color = "red",
        linewidth = 1) +
labs(title = "Nonlinear Regression: Y = (aX) / (b+X)",
x = "X", y = "Y") +
theme_minimal()

print(p)
```

Listing 2: Non linear regression code in R

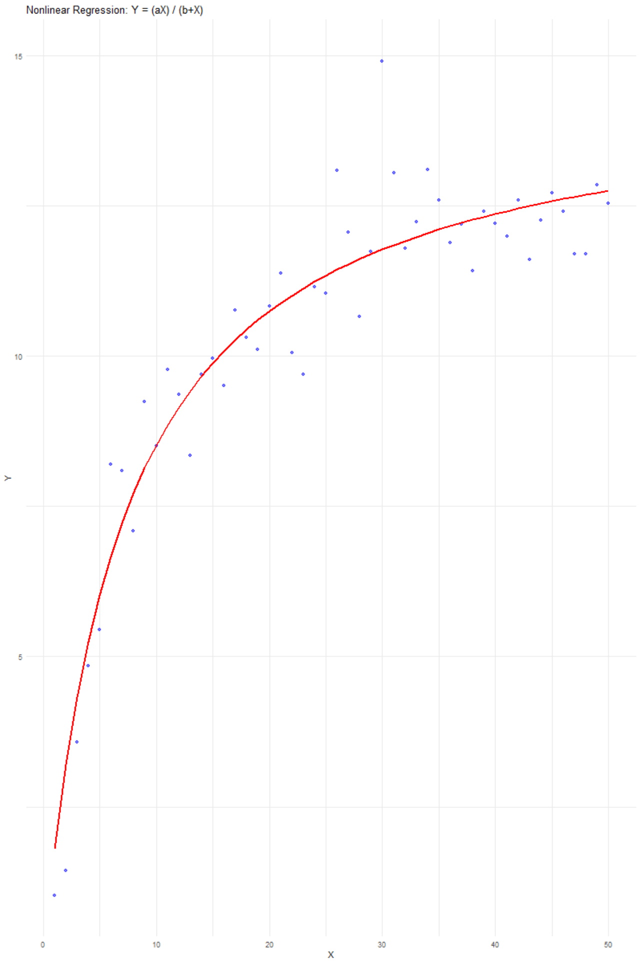


Figure 4: Non-linear regression fit: observed vs. predicted values

4.1 Construct a computer code in R to find the Jackknife and Bootstrap estimators of a and b. In the case of Jackknife, section randomly the sampling into 5 partitions of size 10. In the case of - Bootstrap, generate 1000 samples of size 100 with replacement.

The code in Listing 3 executes the following steps on the data to implement Jackknife resampling with non linear regression. The resulting plots are presented in Figure 5. The estimated parameters are $\hat{a}_{jk} = 14.504$ and $\hat{b}_{jk} = 6.996$. The code executes the following steps to estimate the parameters:

1. **Shuffle the dataset:** We randomly shuffle the data to remove any or-

dering bias:

2. **Divide the data into m Jackknife partitions:** We split the dataset into $m = 5$ partitions, each missing a unique subset of 5 elements.

3. **Fit the NLS model for each Jackknife sample:** We fit a non-linear regression model to each Jackknife sample using Nonlinear Least Squares (NLS) to estimate parameters a and b. The model is defined as:

$$Y = \frac{aX}{b + X} \tag{12}$$

   and is refitted for each sample S_{-a}, which excludes partition P_a.

4. **Compute Jackknife bias-corrected estimates:** The Jackknife estimate for each parameter is calculated using the bias correction formula:

$$\hat{\theta}_{\text{jack}} = m\hat{\theta} - (m - 1)\hat{\theta}_{(-a)} \tag{13}$$

   where:

   - $m = 5$ is the number of partitions.
   - $\hat{\theta}$ is the parameter estimate from the full dataset.
   - $\hat{\theta}_{(-a)}$ is the parameter estimate from the jackknife sample with partition a removed.

5. **Compute final Jackknife estimates for a and b:** The final Jackknife estimates for a and b are obtained by averaging the bias-corrected values across all jackknife samples:

$$\hat{a}_{jk} = \frac{1}{m} \sum_{a=1}^{m} \hat{a}_{\text{jack},a}, \quad \hat{b}_{jk} = \frac{1}{m} \sum_{a=1}^{m} \hat{b}_{\text{jack},a} \tag{14}$$

```
partition_size <- 5

#shuffle the dataframe
set.seed(123)
df_shuf <- df[sample(nrow(df)), ]

# Generate jackknife samples by removing each fold of 5 elements
# lapply applies a function to each element of a list
# my list is from 1:5
# the function will
# for a = 1 remove element 1 to 5 (5*(1-1)+1):(5*1)
# and so on
# note the - sign -- I am removing the elements
# so the return for each a is y without the elements 1 to 5, 6 to 10, etc.
jackknife_samples <- lapply(1:partition_size,
function(a) df_shuf[-((partition_size * (a - 1) + 1):(partition_size * a)), ])

# we have calculated theta_hat_m before
theta_hat_m <- estimated_params

theta_m_a <- function(data) {
        model <- nls(y ~ (a * x) / (b + x),
        data = data,
        start = list(a = theta_hat_m["a"],
        b = theta_hat_m["b"]))
        return(coef(model)) }

# jackknife estimator for each partition
nlsjk <- sapply(jackknife_samples, function(y_a) partition_size * theta_hat_m - (partition_size - 1) * theta_m_a(y_a))

#evaluating the jackknife estimator of the parametrs
jackknife_estimates <- rowMeans(nlsjk)

a_hat_jk <- jackknife_estimates["a"]
b_hat_jk <- jackknife_estimates["b"]

df$predicted_jk <- (a_hat_jk * df$x) / (b_hat_jk + df$x)

# Plot with Jackknife predictions and legend
p_jk <- ggplot(df, aes(x = x, y = y)) +
geom_point(color = "blue", alpha = 0.5, size = 3) +
geom_line(aes(y = Predicted, color = "Full Sample Prediction"),
linewidth = 1, linetype = "dashed") +
geom_line(aes(y = predicted_jk, color = "Jackknife Prediction"),
linewidth = 1) +
labs(title = "Nonlinear Regression: Full Sample vs. Jackknife",
x = "X", y = "Y", color = "Legend") +
theme_minimal() +
scale_color_manual(values = c("Full Sample Prediction" = "red",
"Jackknife Prediction" = "green"))
```

Listing 3: Jackknife resampling code in R

The resulting plot is shown below in Figure 5

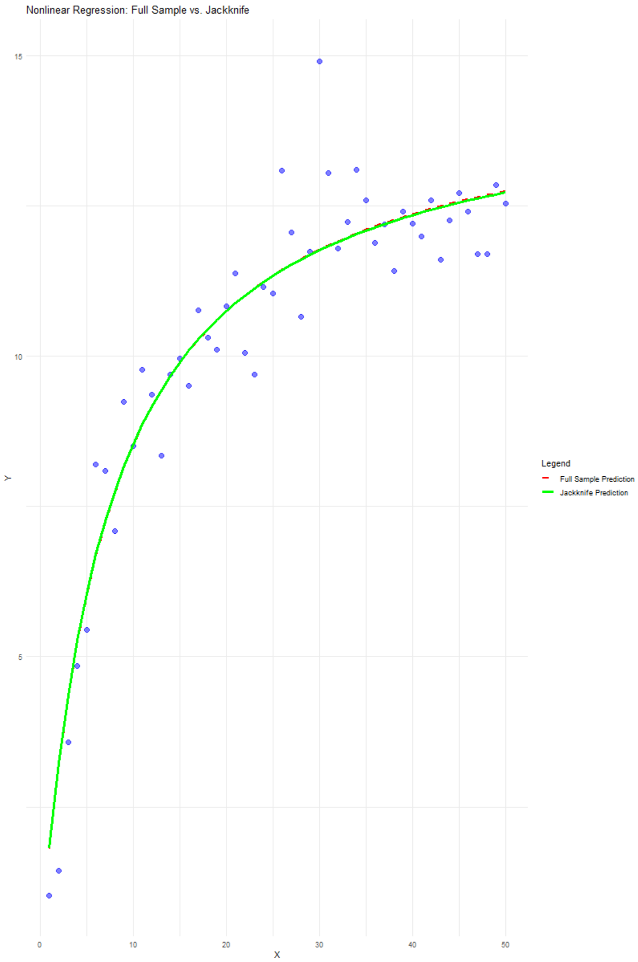


Figure 5: Nonlinear regression fit: observed vs. predicted values including Jackknife predictions

The code in Listing 4 executes the following steps on the data to implement Bootstrap resampling with non linear regression. The resulting plots are presented in Figure 6. The estimated parameters are $\hat{a}_{bs} = 14.566$ and $\hat{b}_{bs} = 7.110$. The code executes the following steps to estimate the parameters:

- **Generate Bootstrap Samples:** WE create 1000 resampled datasets of size 100 by drawing with replacement from the 50 original observations.
- **Fit a Nonlinear Regression Model:** For each bootstrap sample, we estimate parameters a and b using Nonlinear Least Squares (NLS) with the model.

- **Compute Bootstrap Estimates:** The final bootstrap estimates are obtained by averaging the parameter estimates from all bootstrap samples:
- **Predict Values Using Bootstrap Estimates:** Using the estimated parameters $\hat{a}_{\text{bs}}, \hat{b}_{\text{bs}}$, we compute the predicted values:

$$\hat{y}_{\text{bs}} = \frac{\hat{a}_{\text{bs}} x}{\hat{b}_{\text{bs}} + x} \tag{15}$$

```r
num_samples <- 1000
sample_size <- 100

# 1000 bootstrap samples of size 100 with replacement
bootstrap_samples <- lapply(1:num_samples, function(i)
    df[sample(nrow(df), sample_size, replace = TRUE), ])

fit_bootstrap_nls <- function(data) {
        model <- nls(y ~ (a * x) / (b + x),
        data = data,
        start = list(a = 1, b = 1))  # Initial guesses
        return(coef(model))
}

# Apply NLS to each bootstrap sample
nlsbs <- lapply(bootstrap_samples, fit_bootstrap_nls)

# Convert list of bootstrap estimates to a matrix
bootstrap_estimates <- do.call(rbind, nlsbs)
colnames(bootstrap_estimates) <- c("a", "b")

# Compute mean estimates for a and b
a_hat_bs <- mean(bootstrap_estimates[, "a"], na.rm = TRUE)
b_hat_bs <- mean(bootstrap_estimates[, "b"], na.rm = TRUE)

# Print results
cat("Bootstrap Estimated a:", a_hat_bs, "\n")
cat("Bootstrap Estimated b:", b_hat_bs, "\n")


df$predicted_bs <- (a_hat_bs * df$x) / (b_hat_bs + df$x)


# Plot with Jackknife predictions and legend
p_bs <- ggplot(df, aes(x = x, y = y)) +
geom_point(color = "blue", alpha = 0.5, size = 3) +
geom_line(aes(y = Predicted, color = "Full Sample Prediction"),
linewidth = 1, linetype = "dashed") +
geom_line(aes(y = predicted_jk, color = "Jackknife Prediction"),
linewidth = 1) +
geom_line(aes(y = predicted_bs, color = "Bootstrap Prediction"),
linewidth = 1) +
labs(title = "Nonlinear Regression: Full Sample vs. Jackknife vs.
    Bootstrap",
x = "X", y = "Y", color = "Legend") +
theme_minimal() +
scale_color_manual(values = c("Full Sample Prediction" = "red",
"Jackknife Prediction" = "green",
"Bootstrap Prediction" = "blue"))
```

Listing 4: Bootstrap resampling code in R

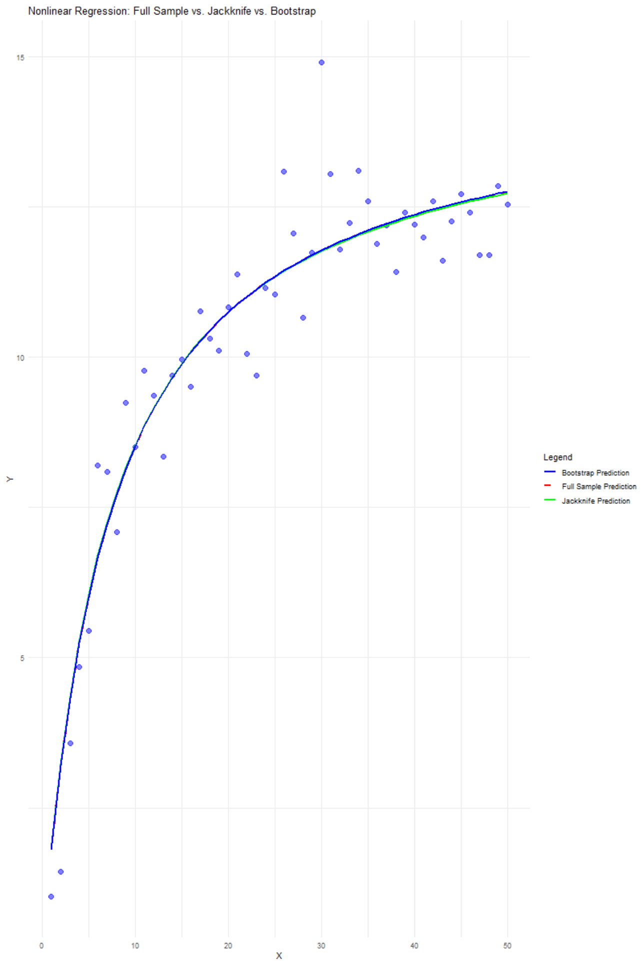


Figure 6: Nonlinear regression fit: observed vs. predicted values including Jackknife and Bootstrap predictions

4.2 For the Jackknife estimator, find a 95% confidence interval using the normal distribution and the t-distribution. For the Bootstrap estimator, find a 95% normal, t and empirical confidence intervals.

The following results were obtained for the Jackknife resampling method:

5 Question 5 – The EM Algorithm

Consider a univariate K-Gaussian mixture model with probability density function:

Method	**Confidence Interval for a**	**Confidence Interval for b**
Jackknife Resampling		
Normal Distribution	`[14.14397, 14.86412]`	`[6.356596, 7.63524]`
t-Distribution	`[13.99397, 15.01412]`	`[6.090267, 7.90157]`
Bootstrap Resampling		
Normal Distribution	`[14.07722, 15.05515]`	`[6.111358, 8.109172]`
t-Distribution	`[14.07663, 15.05575]`	`[6.110146, 8.110384]`
Empirical Distribution	`[14.08458, 15.08578]`	`[6.132766, 8.113393]`

Table 2: Confidence Intervals for Parameters a and b Using Jackknife and Bootstrap Methods

$$f(x) = \sum_{l=1}^{K} \pi_l \phi(x, \mu_l, \sigma_l)$$

such that $\sum_{l=1}^{K} \pi_l = 1$ and $\pi_l > 0$ for all l, and where $\phi(x, \mu, \sigma)$ is the Gaussian density function. The EM algorithm for this works as follows:

1. Initialise $\mu_1^{(0)}, \ldots, \mu_K^{(0)}, \sigma_1^{(0)}, \ldots, \sigma_K^{(0)}, \pi_1^{(0)}, \ldots, \pi_K^{(0)}$.

2. Let
$$\gamma_n^{(j,k)} = \frac{\pi_k \phi(x_n | \mu_k^{(j-1)}, \sigma_k^{(j-1)})}{\sum_{i=1}^{K} \pi_i \phi(x_n | \mu_i^{(j-1)}, \sigma_i^{(j-1)})}.$$

3. Let
$$\mu_k^{(J)} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_n^{(j,k)} x_n,$$
$$\sigma_k^{(J)} = \sqrt{\frac{1}{N_k} \sum_{n=1}^{N} \gamma_n^{(j,k)} (x_n - \mu_k^{(J)})^2},$$
and
$$\pi_k^{(J)} = \frac{N_{jk}}{N},$$
where
$$N_{jk} = \sum_{n=1}^{N} \gamma_n^{(j,k)}.$$

5.1 Simulate 1000 readings from a mixture Gaussian distribution with 3 or more Gaussians.

The following function generalizes the code provided in the question so that an arbitrary number of samples are generated from any number of Gaussian distributions. The function also lists the number of samples selected from each

Gaussian to ensure that acceptable proportions were attained for each sample and plots a distribution of the data and plots of the Gaussians.

```
library(ggplot2)

generate_samples <- function(
mixing_coefficients,
means,
standard_deviations,
number_of_samples = 1000) {

        # should check that lengths are equal

        data <- c()
        choices_count <- rep(0, length(mixing_coefficients))

        # select one of the gaussian distributions according
        # to the mixing coefficients
        choices <- sample(
        x = seq_along(mixing_coefficients),
        size = number_of_samples,
        prob = mixing_coefficients,
        replace = TRUE)

        # let us see that the number of selections make sense
        for (i in 1:number_of_samples){
                choices_count[choices[i]] <-
                    choices_count[choices[i]] + 1
        }
        print(choices_count)

        # for each selection, sample from the gaussian
        data <- c(
        data,
        rnorm(
        n = number_of_samples,
        mean = means[choices],
        sd = standard_deviations[choices]))

        # plot histogram and gaussian curves
        df <- data.frame(data)

        p <- ggplot(df, aes(x = data)) +
        geom_histogram(
        aes(y = ..density..),
        bins = 30,
        fill = "blue",
        alpha = 0.6) +
        stat_function(fun = function(x) {
                Reduce(‘+‘, lapply(1:length(means), function(i) {
                        dnorm(
                        x = x,
                        mean = means[i],
                        sd = standard_deviations[i]) *
                            mixing_coefficients[i]
                }))
        }, color = "red") +
        labs(title = "Histogram of Mixture of Gaussians",
        x = "Value",
        y = "Density") +
        theme_bw()

        print(p)
}
```

Listing 5: Gaussian mixture sample generation code in R

Data from the mixed Gaussian distributions was subsequently generated using the parameters below. The plot that was obtained as a result of this process is shown in Figure 7

$\pi_1 = 0.2$	$\mu_1 = 6$	$\sigma_1 = 2.0$
$\pi_2 = 0.5$	$\mu_2 = 0$	$\sigma_2 = 1.0$
$\pi_3 = 0.3$	$\mu_3 = -7$	$\sigma_3 = 1.5$

```
mixing_coefficients <- c(0.2, 0.5, 0.3)
means <- c(6, 0, -7)
standard_deviations <- c(2, 1, 1.5)

generate_samples(mixing_coefficients, means,
    standard_deviations)
```

Listing 6: Generation of data for this question

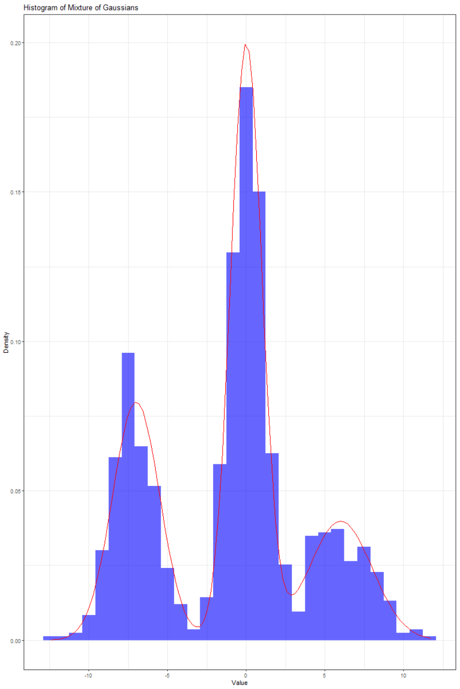


Figure 7: Distribution of generated data

5.2 Determine initial values $\mu_1^{(0)}, \ldots, \mu_K^{(0)}, \sigma_1^{(0)}, \ldots, \sigma_K^{(0)}, \pi_1^{(0)}, \ldots, \pi_K^{(0)}$ using a K-means clustering approach or otherwise.

The function in Listing 7 is a custom implementation of a k-means algorithm to initialize parameters($\mu^{(0)}$, $\sigma^{(0)}$, and $\pi^{(0)}$) for a Gaussian mixture model. Its salient features are the following:

- After the initial centroids are randomly selected, the k-means algorithm iteratively assigns the data point to the closest centroid based on the absolute distance. The cluster centroids are then updated as the means of the points assigned to each cluster until the change in centroids is small or the threshold number of iterations is reached.
- The means of each centroid is, as we have discussed above, the centroid of each cluster.
- the standard deviation is calculated as follows

  $$\sigma_j = \begin{cases} \sqrt{\frac{1}{n_j} \sum_{x_i \in C_j} (x_i - \mu_j)^2}, & \text{if } n_j > 0 \\ 0, & \text{otherwise} \end{cases}$$

  Where:

  - σ_j is the standard deviation for cluster j,
  - C_j is the set of points in cluster j,
  - n_j is the number of points in cluster j,
  - x_i are the individual data points in cluster j,
  - μ_j is the centroid (mean) of cluster j,
- The initial values for the mixing coefficients of the Gaussian Mixture Model are computed as:

  $$\pi_j = \begin{cases} \frac{n_j}{N}, & \text{if } n_j > 0 \\ 0, & \text{otherwise} \end{cases}$$

  Where:

  - π_j is the mixing coefficient for cluster j,
  - n_j is the number of data points in cluster j,
  - N is the total number of data points in the dataset,

```r
initial_values_knn <- function(data, k = 3) {
        set.seed(42)  # For reproducibility

        # Randomly select k initial centroids
        centroids <- sample(x = data, size = k, replace = FALSE)

        # Initialize empty clusters
        clusters <- vector(mode = "list", length = k)

        for (iteration in 1:100) {
                # Reset clusters
                clusters <- vector(mode = "list", length = k)

                # Assign each data point to the closest centroid
                for (item in data) {
                        distances <- abs(item - centroids)
                        closest_centroid <- which.min(distances)
                        clusters[[closest_centroid]] <- c(clusters[[closest_centroid]], item)
                }

                # Compute new centroids
                new_centroids <- sapply(1:k, function(i) {
                        if (length(clusters[[i]]) > 0) {
                                mean(clusters[[i]])
                        } else {
                                centroids[i]  # Keep old centroid if no points are assigned
                        }
                })

                # Check for convergence
                if (max(abs(new_centroids - centroids)) < 0.0001) {
                        break
                }

                centroids <- new_centroids
        }

        # Compute standard deviations
        clusters_standard_devs <- sapply(1:k, function(cluster_idx) {
                if (length(clusters[[cluster_idx]]) > 0) {
                        sqrt(sum((clusters[[cluster_idx]] - centroids[cluster_idx])^2) / length(clusters[[cluster_idx]]))
                } else {
                        0
                }
        })

        # Compute mixing coefficients for each cluster
        mixing_coefficients <- sapply(1:k, function(cluster_idx) {
                length(clusters[[cluster_idx]]) / length(data)
        })

        return(list(
        centroids = centroids,
        standard_devs = clusters_standard_devs,
        mixing_coefficients = mixing_coefficients))
}
```

Listing 7: k-means algorithm to compute the initial values of the mixed Gaussian model parameters

The results obtained from this function are as follows:

$\pi_1 = 0.179$	$\mu_1 = 6.353$	$\sigma_1 = 1.685$
$\pi_2 = 0.545$	$\mu_2 = 0.104$	$\sigma_2 = 1.107$
$\pi_3 = 0.276$	$\mu_3 = -6.942$	$\sigma_3 = 1.508$

Table 3: Estimated parameters for the initial values of Gaussian mixture model

5.3 Run the EM algorithm for a number of iterations. Print $\mu_1^{(j)}, \dots, \mu_k^{(j)}, \sigma_1^{(j)}, \dots, \sigma_k^{(j)}, \pi_1^{(j)}, \dots, \pi_k^{(j)}$ for each iteration, and also the log-likelihood, which is given by: $\sum_{n=1}^{N} \ln \left(\sum_{l=1}^{K} \phi(x_i|\mu_l^{(j)}, \sigma_l^{(j)}) \right)$. Determine when to stop the EM algorithm, either via a maximum number of iterations or through a convergence criterion. However, ensure that the EM algorithm has converged. Plot the trajectory of the estimates and the likelihood by iteration to illustrate this.

We start by examining the code created to execute this question, which is presented below in four listings.

- Listing 8 contains the estimation step code that computes the responsibility of each Gaussian component for each data point as:

$$\gamma_{n,k} = \frac{\pi_k \phi(x_n|\mu_k, \sigma_k)}{\sum_{j=1}^{K} \pi_j \phi(x_n|\mu_j, \sigma_j)}$$

  Where:

  - $\gamma_{n,k}$ is the responsibility of Gaussian k for data point x_n.
  - π_k is the mixing coefficient for Gaussian k.
  - $\phi(x_n|\mu_k, \sigma_k)$ is the Gaussian probability density function:

$$\phi(x_n|\mu_k, \sigma_k) = \frac{1}{\sigma_k \sqrt{2\pi}} \exp\left(-\frac{(x_n - \mu_k)^2}{2\sigma_k^2} \right).$$

- Listing 9 illustrates the maximization step of the EM algorithm that updates the parameters as follows:

  - We update the mixing coefficients to calculate the proportion of data points that belong to a given Gaussian distribution:

$$\pi_k^{(j)} = \frac{N_k^{(j)}}{N}$$

Where:

$$N_k^{(j)} = \sum_{n=1}^{N} \gamma_{n,k}^{(j)}$$

Is the total responsibility weight assigned to Gaussian k?

- We update the means to compute the weighted mean of the data points assigned to each Gaussian:

$$\mu_k^{(j)} = \frac{\sum_{n=1}^{N} \gamma_{n,k}^{(j)} x_n}{N_k^{(j)}}$$

- We update the standard deviations to calculate the spread of the data points around the updated mean for each Gaussian:

$$\sigma_k^{(j)} = \sqrt{\frac{\sum_{n=1}^{N} \gamma_{n,k}^{(j)} (x_n - \mu_k^{(j)})^2}{N_k^{(j)}}}$$

- We then calculate the log-likelihood function for our Gaussian mixture model using the code in Listing 10 using the suggested formulation:

$$\mathcal{L} = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \phi(x_n | \mu_k, \sigma_k) \right)$$

Where:

- $\mathcal{L}$ is the log-likelihood of the data.
- K is the number of Gaussian components.
- N is the number of data points.
- π_k is the mixing coefficient for Gaussian k.
- $\phi(x_n | \mu_k, \sigma_k)$ is the Gaussian probability density function:

$$\phi(x_n | \mu_k, \sigma_k) = \frac{1}{\sigma_k \sqrt{2\pi}} \exp \left(-\frac{(x_n - \mu_k)^2}{2\sigma_k^2} \right)$$

- Listing 11 shows the execution of the EM algorithm, starting from the generation of data to the estimation of the initial values and ending with the actual estimates using the EM algorithm. We notice that EM is allowed a maximum of execution 100 steps, but optimization will stop if the change in log-likelihood is smaller than 10^{-5}.

```
expectation_step <- function(data, means, standard_deviations, mixing_coefficients) {

        number_of_gaussians <- length(mixing_coefficients)

        # Initialize gamma matrix
        gamma <- matrix(0, nrow = length(data), ncol = number_of_gaussians)

        # Calculate denominator (total probability for each data point)
        den_total <- 0
        for (k in 1:number_of_gaussians) {
                den_total <- den_total + mixing_coefficients[k] * dnorm(data, mean = means[k], sd = standard_deviations[k])
        }

        # Calculate numerator and compute gamma
        for (k in 1:number_of_gaussians) {
                gamma[, k] <- (mixing_coefficients[k] * dnorm(data, mean = means[k], sd = standard_deviations[k])) / den_total
        }

        return(gamma)
}
```

Listing 8: Estimation step code in R

```

maximization_step <- function(data, gamma, means, standard_deviations, mixing_coefficients) {

        no_gaussians <- length(mixing_coefficients)
        m <- length(data)

        # Compute cluster responsibilities
        m_c <- colSums(gamma)  # Sum of responsibilities for each Gaussian

        # Compute new mixing coefficients
        new_mixing_coefficients <- m_c / m

        # Initialize new means and standard deviations
        new_means <- numeric(no_gaussians)
        new_standard_deviations <- numeric(no_gaussians)

        # Compute new means and standard deviations for each Gaussian
        for (k in 1:no_gaussians) {
                new_means[k] <- sum(gamma[, k] * data) / m_c[k]
                new_standard_deviations[k] <- sqrt(sum(gamma[, k] * (data - means[k])^2) / m_c[k])
        }

        return(list(
        means = new_means,
        standard_devs = new_standard_deviations,
        mixing_coefficients = new_mixing_coefficients
        ))
}
```

Listing 9: Maximization step code in R

```r
log_likelihood <- function(data, means, standard_deviations, mixing_coefficients) {

        number_of_gaussians <- length(mixing_coefficients)

        # Initialize likelihood matrix
        likelihood <- matrix(0, nrow = length(data), ncol = number_of_gaussians)

        # Compute likelihood for each Gaussian component
        for (k in 1:number_of_gaussians) {
                likelihood[, k] <- mixing_coefficients[k] * dnorm(data, mean = means[k], sd = standard_deviations[k])
        }

        # Compute the log-likelihood
        log_likelihood_value <- sum(log(rowSums(likelihood)))

        return(log_likelihood_value)
}
```

Listing 10: Log-likelihood code in R

```
mixing_coefficients <- c(0.2, 0.5, 0.3)
means <- c(6, 0, -7)
standard_deviations <- c(2, 1, 1.5)

# generate the data
data <- generate_samples(mixing_coefficients, means, standard_deviations)

# initial values
kmeans_ret <- initial_values_kmeans(data)

centroids <- kmeans_ret$centroids
standard_devs <- kmeans_ret$standard_devs
mixing_coefficients <- kmeans_ret$mixing_coefficients

print("initial values")
print(centroids)
print(standard_devs)
print(mixing_coefficients)

log_likelihoods <- c()
for (i in 1:100) {
        gamma <- expectation_step(data, centroids, standard_devs, mixing_coefficients)

        max_ret <- maximization_step(data, gamma, centroids, standard_devs, mixing_coefficients)

        centroids <- max_ret$means
        standard_devs <- max_ret$standard_devs
        mixing_coefficients <- max_ret$mixing_coefficients

        ll <- log_likelihood(data, means, standard_devs, mixing_coefficients)

        log_likelihoods <- c(log_likelihoods, ll)

        if (i>2 &&  abs(ll - log_likelihoods[i-1]) < 1e-5) {
                print("converged")
                break
        }
        # form string
        iteration_result_str <- paste( "iteration: ", i)
        for (j in 1:length(centroids)) {
                iteration_result_str <- paste(iteration_result_str, " c",j,": ", format(centroids[j],3))
                iteration_result_str <- paste(iteration_result_str, " s",j,": ", format(standard_devs[j],3))
                iteration_result_str <- paste(iteration_result_str, " m",j,": ", format(mixing_coefficients[j],3))
        }
        print(iteration_result_str)
}
```

Listing 11: Execution code for this question

The parameters obtained after each iteration are shown in Table 4. The algorithm converged after 28 iterations. A plot of the log likelihood at each iteration can be observed in Figure 8

Step	π_1	σ_1	μ_1	π_2	σ_2	μ_2	π_3	σ_3	μ_3	$\mathcal{L}$
1	6.259	1.780	0.183	0.088	1.101	0.541	-6.936	1.519	0.276	-2666.447
2	6.208	1.824	0.186	0.078	1.090	0.538	-6.935	1.520	0.276	-2665.715
3	6.176	1.850	0.187	0.071	1.084	0.537	-6.935	1.521	0.276	-2665.371
4	6.155	1.868	0.188	0.067	1.079	0.536	-6.934	1.521	0.276	-2665.192
5	6.142	1.880	0.189	0.065	1.077	0.535	-6.934	1.521	0.276	-2665.092
6	6.132	1.888	0.189	0.063	1.075	0.534	-6.934	1.522	0.276	-2665.033
7	6.126	1.893	0.189	0.062	1.074	0.534	-6.934	1.522	0.276	-2664.996
8	6.122	1.897	0.190	0.061	1.073	0.534	-6.934	1.522	0.276	-2664.973
9	6.119	1.900	0.190	0.061	1.073	0.534	-6.934	1.522	0.276	-2664.958
10	6.117	1.901	0.190	0.061	1.072	0.534	-6.934	1.522	0.276	-2664.947
11	6.115	1.903	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.941
12	6.114	1.903	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.936
13	6.114	1.904	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.933
14	6.113	1.904	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.931
15	6.113	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.929
16	6.113	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.928
17	6.113	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.928
18	6.113	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.927
19	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.927
20	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.927
21	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926
22	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926
23	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926
24	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926
25	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926
26	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926
27	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926
28	6.112	1.905	0.190	0.060	1.072	0.534	-6.934	1.522	0.276	-2664.926

Table 4: EM iterations results table

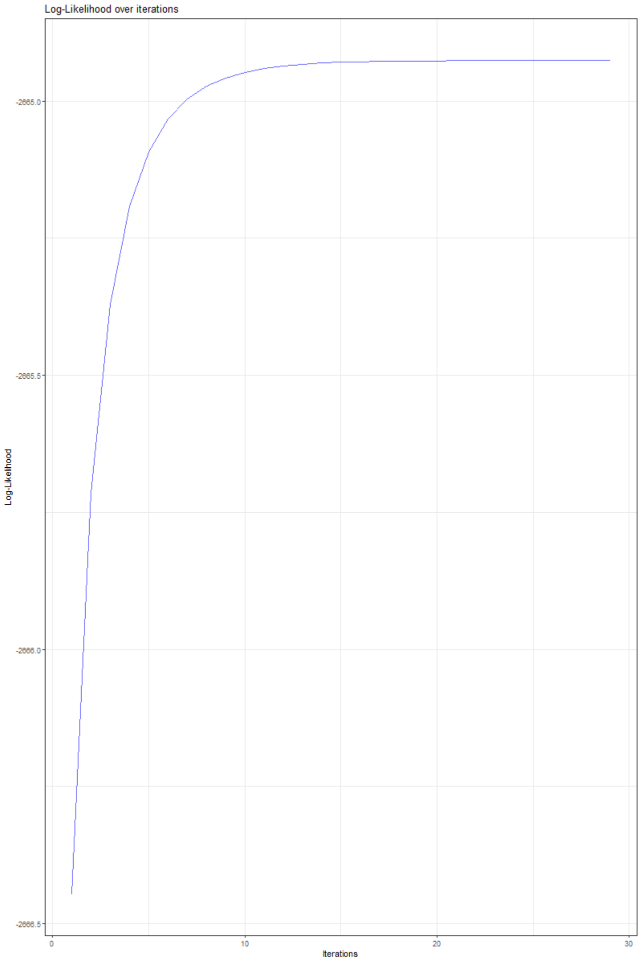


Figure 8: Log-likelihood vs iterations for the execution of the EM algorithm until convergence

5.4 Give the final estimated μ_k's, σ_k's and π_k's. How do these compare with the coefficients you chose in the simulation?

The final estimated values compared to the chosen ones are listed in Table 5. The estimated values are very close to the initially chosen values, confirming that the initial assumptions were reasonable. The final log-likelihood value of -2664.926 indicates that the EM algorithm successfully converged to a stable solution. The minor parameter adjustments demonstrate that the algorithm effectively fine-tuned the initial estimates based on the data. The estimated values align closely with the assumed mixture structure, validating the underlying data distribution. Minor shifts in mixing coefficients, means, and standard deviations suggest that while the clusters generally adhered to the expected patterns, some exhibited slightly different densities than initially assumed. The successful convergence of the model indicates that the algorithm reached a local maximum of the likelihood function, further confirming the robustness of the estimation process.

Distribution	Parameter	Chosen Value	Estimated Value
Distribution 1	π_1	0.200000	0.1901318
	μ_1	6.000000	6.112331
	σ_1	2.000000	1.905292
Distribution 2	π_2	0.500000	0.5335142
	μ_2	0.000000	0.05972212
	σ_2	1.000000	1.071750
Distribution 3	π_3	0.300000	0.276354
	μ_3	-7.000000	-6.933639
	σ_3	1.500000	1.521854

Table 5: Chosen vs. estimated parameters

References

[1] Jalal Mahmud, Jilin Chen, and Jeffrey Nichols. When will you answer this? estimating response time in twitter. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 7, pages 697–700, 2013.