A Report
On
Modeling EEG Signals Using Polynomial Regression

By:Aayush Adhikari

220426@softwarica.edu.np

Contents

Coursework Description:

In this assignment, we have to s work on the best regression model which can describe the relationship between brain electroencephalogram signals. Compared with other neuroimage data, such as functional magnetic resonance imaging (fMRI), the EEG technique is much cheaper and has an excellent temporal resolution. As a result, EEG-based analysis and modeling approaches have been extensively applied to characterize various neurological disorders (e.g. Parkinson's disease, epilepsy, tremor) and the development of the brain-computer interface. To achieve those goals, it is very important to understand the connectivity between different brain areas, which can be obtained through the modeling and analysis of different EEG channels.

Introduction to EEG signals

Electroencephalography signals are the signatures of neural activities and generally are the integrals of action potentials that elicit from the brain with different latencies and populations around each time instant. Modeling neural activities is probably more difficult than modeling the function of any other organ. The application of time–frequency (TF) domain analysis for detection of the seizure in neonates has paved the way for further research in this area. An EEG signal can be considered the output of a nonlinear system, which may be characterized deterministically. Nonstationarity of the signals can be quantified by evaluating the changes in signal distribution over time. The synchro-squeezing wavelet transform has been introduced as a post-processing technique to enhance the TF spectrum obtained by applying the wavelet transform. Empirical mode decomposition is an adaptive time–space analysis method suitable for processing nonstationary and nonlinear time series.

1. Analysis of preliminary data

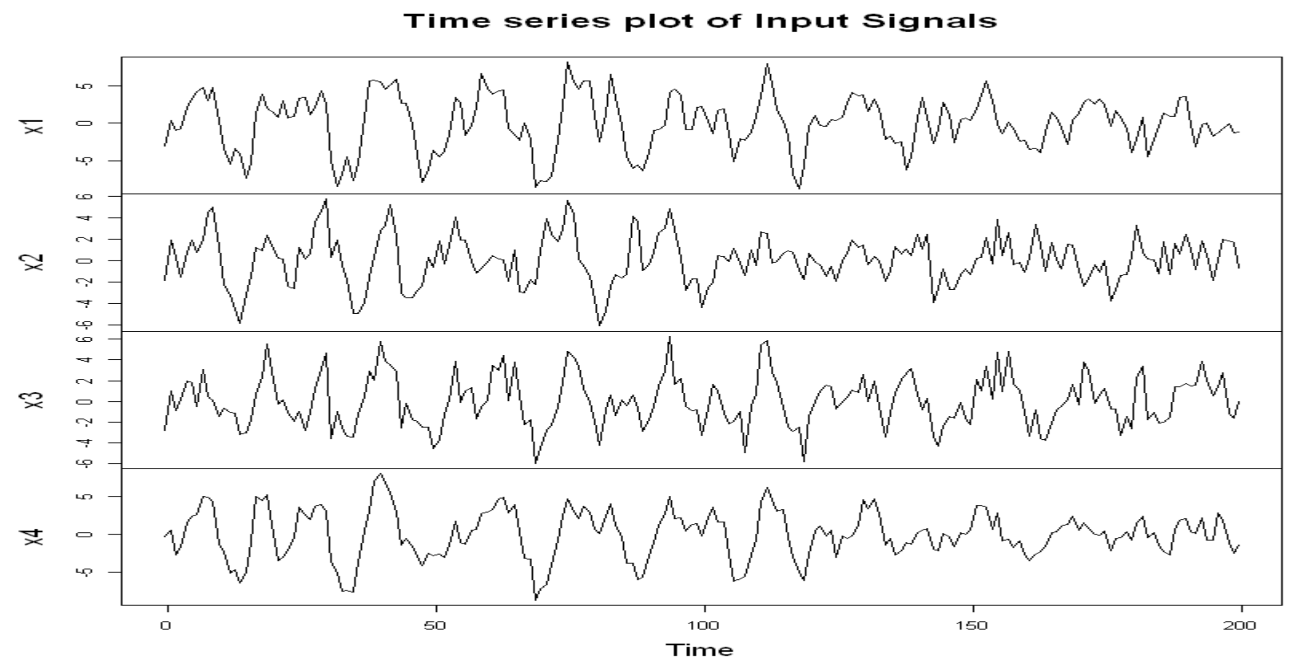
The main objective of preliminary data analysis is to edit, clean the data and prepare for further analysis. It describes the features and summarizes the result. We use qualitative and quantitative analysis to find our objective. In this process, we use scales of measurement, types of data, graphical methods of analysis including histograms, probability plots, and other graphical representations of data, and basic descriptive statistics, mean, median, standard deviation, and so forth.

Plots of Input and Output Time Series Plot of EEG Signals

A graph that displays data collected in a time sequence from any process is called a time series plot. It shows observations against time. They are alike to the x-y graphs, but x-y graphs can graph any type of x variables, whereas the time series plot can only graph time variables on the x-axis. These plots do not have any categories like pie charts and bar charts. These plots show the changes in data over time.

The aim is to compute the pattern of the data and predict the future based on this data. The data in the time series plot are not random. It consists of the data of the consecutive experiments. The temperature of a particular city recorded at noon for 30 consecutive days is an example of time series.

The plot calms down and displays some recurrent patterns after 120ms.. Out of the four signals, X4 appears to be the most stable, with fewer spikes and a more recognizable pattern. The spikes are scarce after the 120 ms limit. The output signal begins gradually up until the 60 ms threshold. We can see the spikes are leveling down and following a steady pattern later.



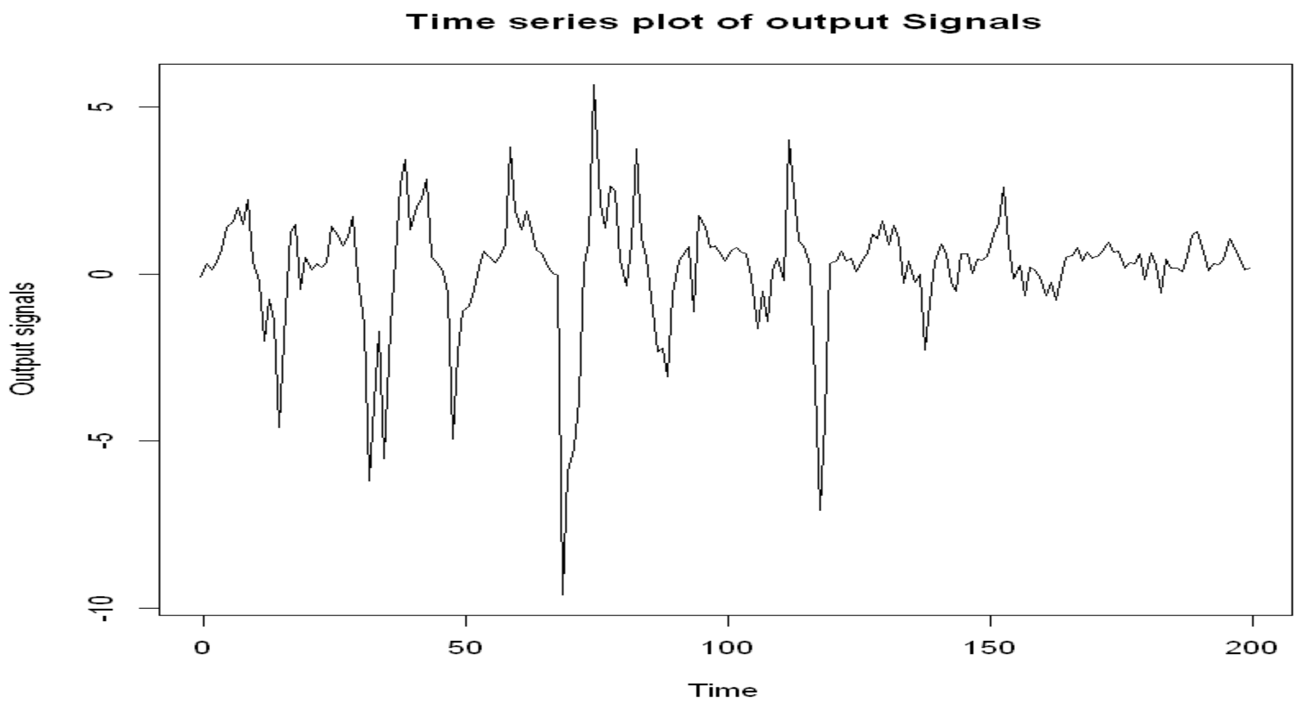
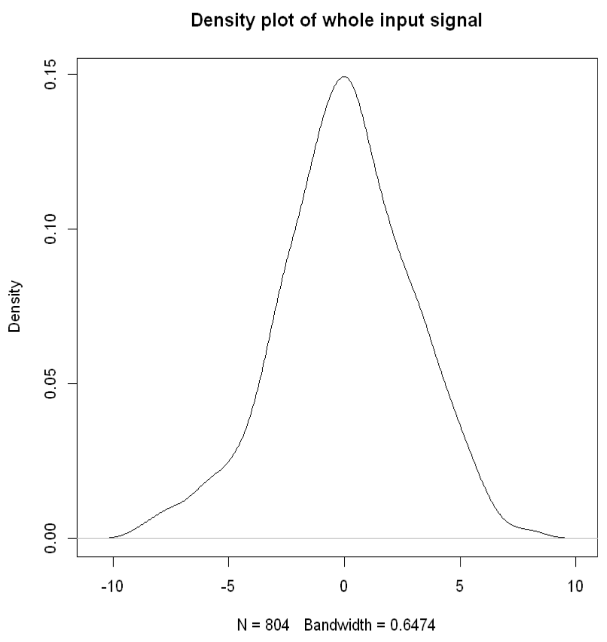


Fig 1: Time series plot of input and output signal

Distribution for each EEG signals:

Distribution plots visually assess the distribution of sample data by comparing the empirical distribution of the data with the theoretical values expected from a specified distribution.

As density plots are better at determining the distribution shape because they’re not affected by the number of bins like in histograms. In order words, we can say density plot is the continuous and smoothed version of the histogram. The choice of the kernel function in a density plot depends on the data's characteristics and the analysis's goals.



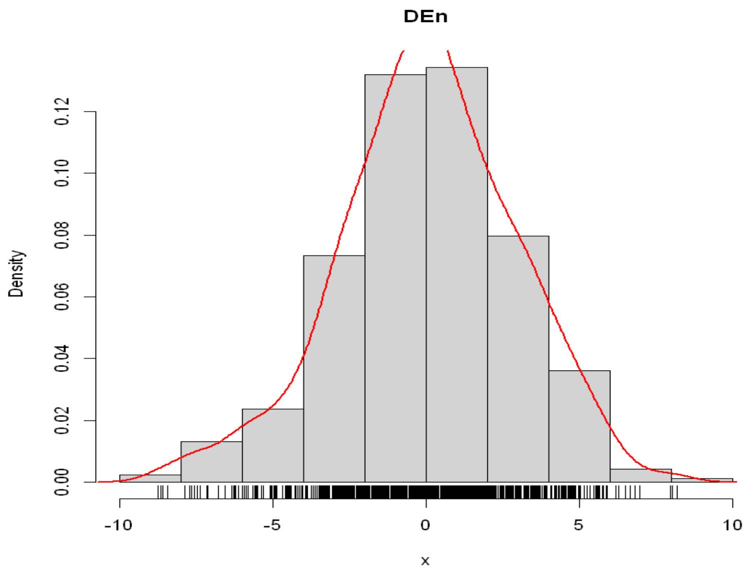
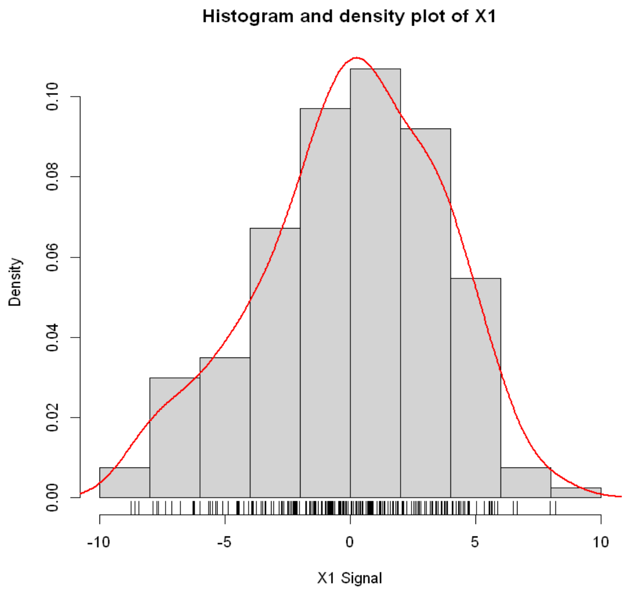


Fig 2: Density Plot and histogram plot of Input signal



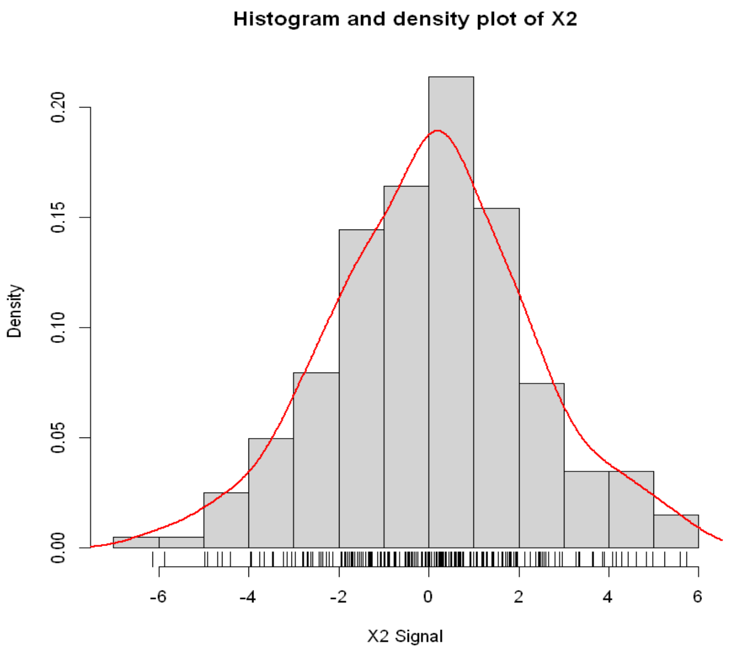


Fig3: Figure of histogram and density plot of X1 and X2 signal

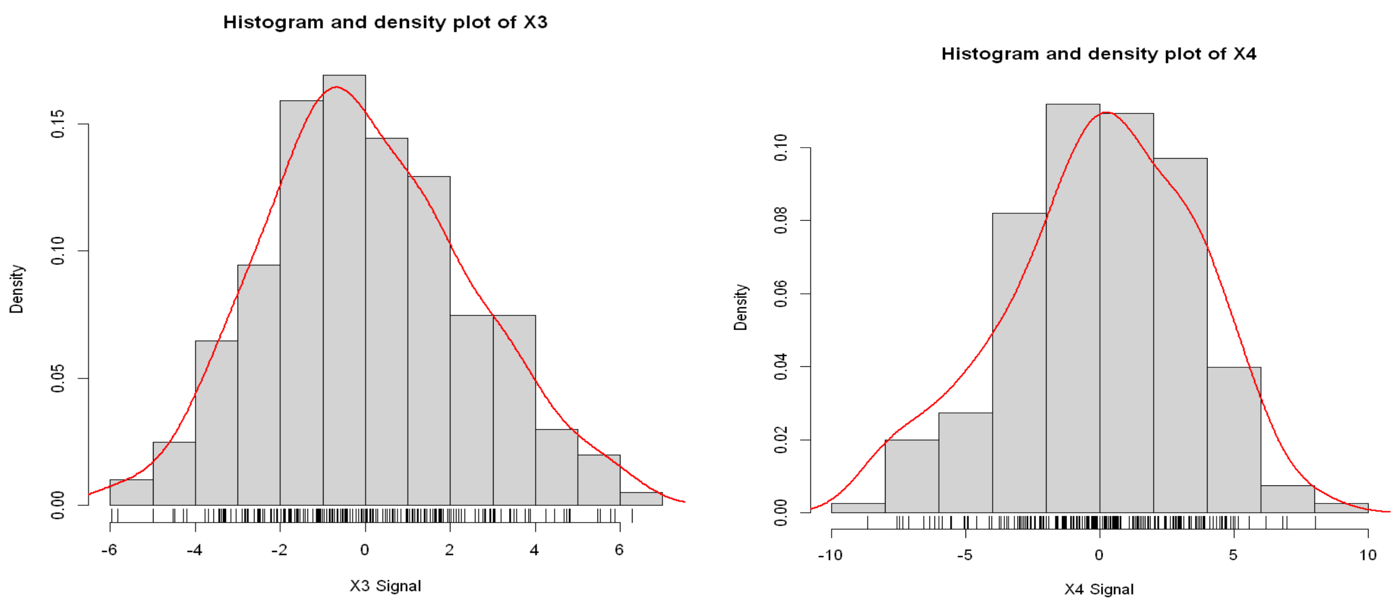


Fig4: Figure of X3 signal Histogram and density plot of X3 and X4 signals

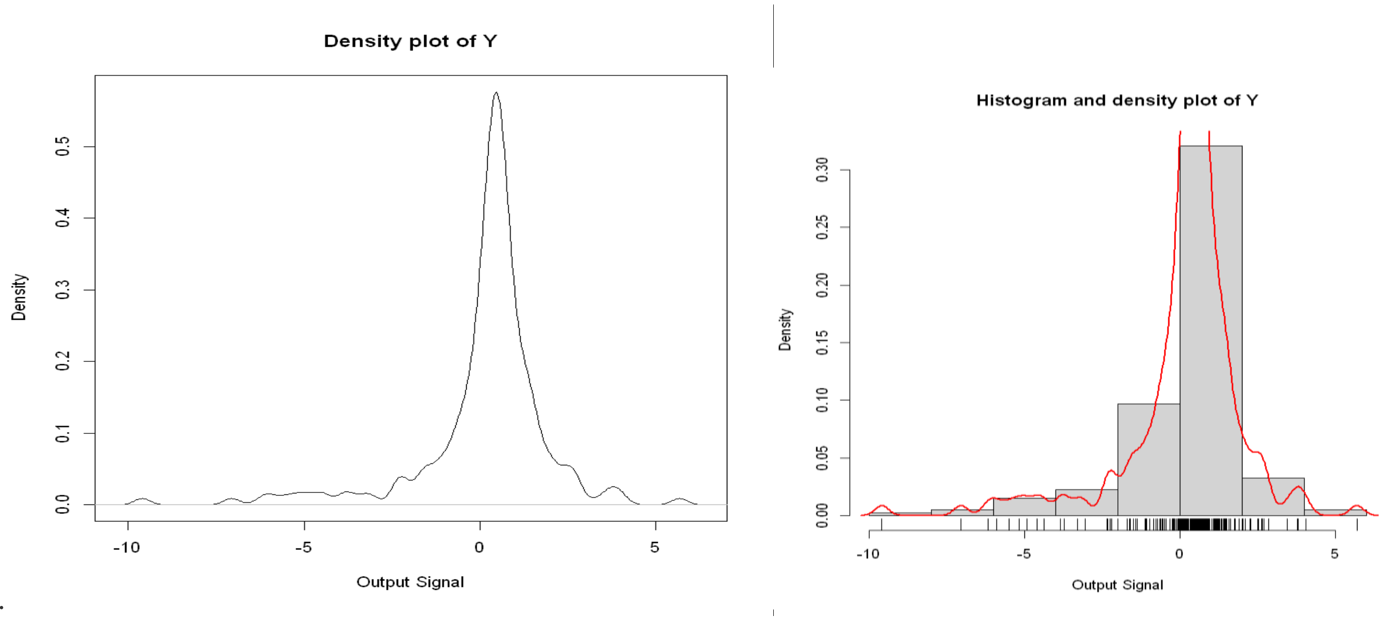


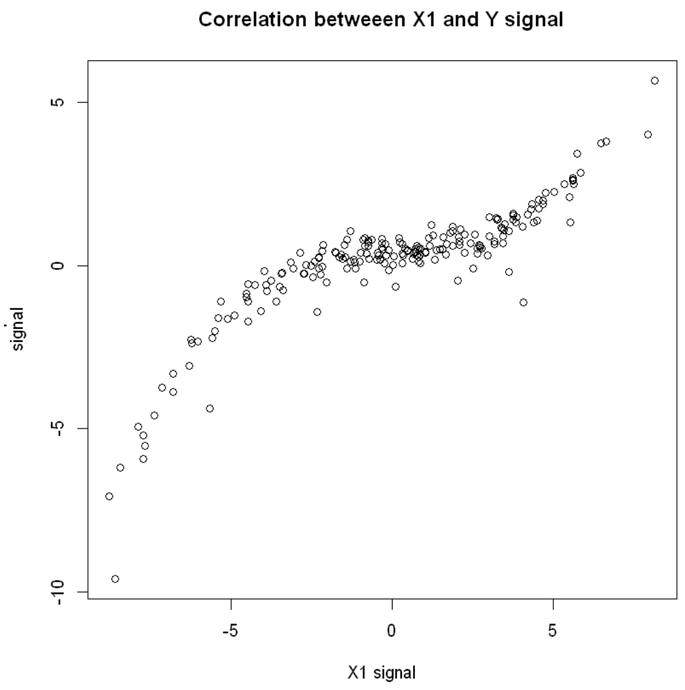
Fig5: Figure of Density plot and density plot with histogram of Output signal

Calculation of correlation and scatter plots to examine their dependencies

The relationship between two quantitative variables measured for the same individuals can be shown by a Scatter plot. The values of one variable appear on the horizontal axis, and the values of the other variable appear on the vertical axis. Each individual in the data appears as a point on the graph. When we add a line of best fit to a scatter plot, we can also see the correlation (positive, negative, or zero) between the two variables

The plot shows the relationship between multiple variables. One main reason to use a correlation plot is when we have a large number of variables, and we want to see how they are related to each other. We will be using 95% of CI. A 95% CI for the mean of a population provides a range of plausible values for the true mean of the population.

Let's now examine the input and output signals of the given data. In the picture below that shows the relationship between the X1, X2, and X3 signals and the Y signal, we can see a pattern where the data appears to follow a low positive correlation since the dot plot has a greater Y-value on the X-axis and the data are not dispersed.



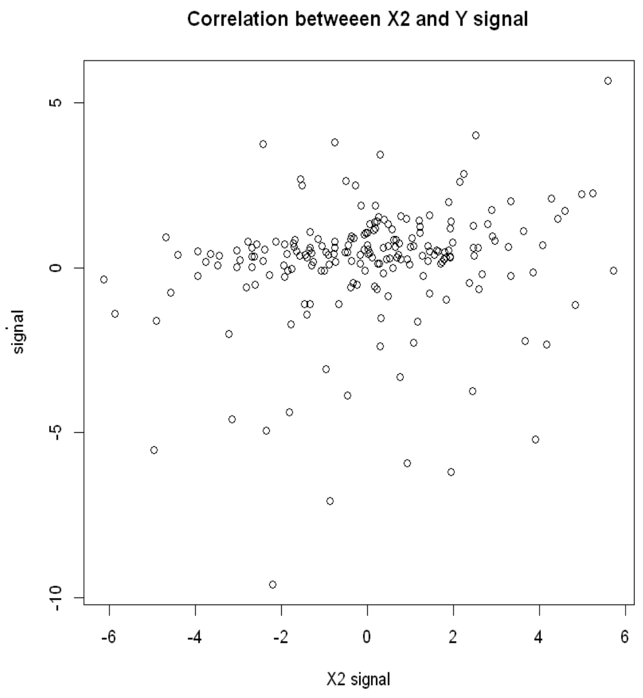
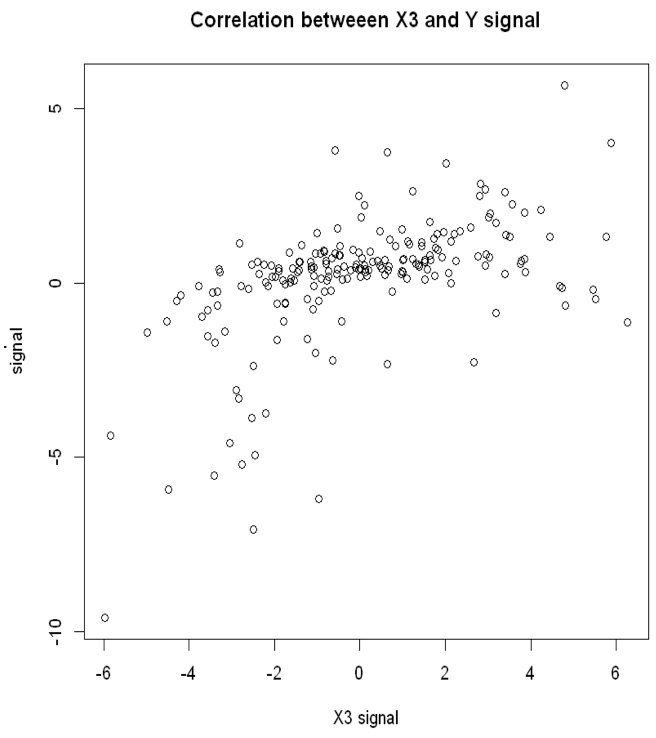


Fig11: Correlation and scatter plot of X1 and X2 signal



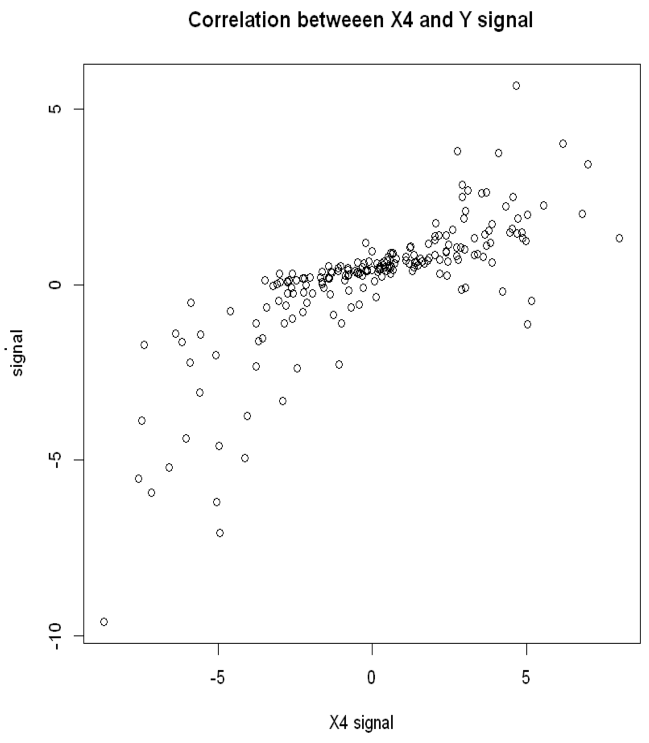


Fig12: Figure of Correlation and scatter plot of X3 and X$ signal

2. Regression modeling of the relationship between EG signals

We have to detect a suitable model with observing the relationship between the input EEG signals and the output signal, There are 5 candidate nonlinear polynomial regression models, and only one of them is suitable. The objective is to identify this ‘true’ model from those candidate models following Tasks 2.1 – 2.6.

Using least square, estimating model parameter

We have to find the best possible fit line of the regression equation or the date point which shows the relationship between an unknown dependent value with the independent value. So Least Square Errors is used in this process. It can be calculated as (X TX) -1X T θ y” .

In R we can use the following code to bind input data ie; x_model1 <- cbind(ones, x[,4], x[,1]^2, x[,1]^3, x[,3]^4). We can then use the least squares formula after formatting input data, as mentioned above and use the built-in solve linear equations function called solve().

Theta Hat for model 1

0.4015807794	0.1277101097	-0.0002902170	0.0096688129	-0.0004098917	-0.0001543367

Table1. Theta Hat of Model1

Theta Hat for model 2

0.483065688	0.143578928	0.010038614	-0.001912836

Table2. Theta Hat of Model2

Theta Hat for model 3

0.340561975	0.021330543	-0.002857744

Table3. Theta Hat of Model3

Theta Hat for model 4

0.509013488	0.053322048	0.012067145	-0.001855997

Table4. Theta Hat of Model4

Theta Hat for model 5

0.4798284629	0.143446074	0.0003254641	0.0100562759	-0.0019198749

Table5. Theta Hat of Model5

Calculation of model residual errors (RSS)

The sum of squared errors of prediction (SSE) is called residual sum of squares. By subtracting the average square of the actual values and the estimated values of the dependent variable, based on the model parameters we can calculate this.

$$\mathbf{RSS = \sum i=1n(yi-xi\theta\hat{})2}$$

where:

n :- the number of observations

yi:- the observed value of the dependent variable for the i-th observation

xi :- the value of the independent variable for the i-th observation

θ^ :-the estimated value of the model parameters

The results obtained are listed below:

Model	RSS Value
Model1	35.3966
Model2	2.1398
Model3	463.3124
Model4	20.2590
Model5	2.1355

Calculation of log-likelihood functions

This method (likelihood method) is a measure of how well a particular model fits the data; It explains how well a parameter (θ) explains the observed data. Likelihoods are often tiny numbers (or large products), making them difficult to graph. Taking the natural (base e) logarithm results in a better graph with large sums instead of products.The log-likelihood function is usually (not always!) easier to optimize.

$$\ln p(D|\hat{\boldsymbol{\theta}}) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\hat{\sigma}^2) - \frac{1}{2\hat{\sigma}^2}\text{RSS}$$

where:

- n :- sample size (here, Y output signal)
- (θ^)2 :- variance of the errors (also known as the residual variance) (Task 2.2)
- ln :- natural logarithm
- ln p(D|θ^)" :- l og-likelihood
- π value i:- 2.27
- RSS :- value from Task 2.2

The results obtained are listed below:

Model	**Likehood**
Model1	-110.6707
Model2	171.3245
Model3	-369.1351
Model4	-54.5899
Model5	171.5247

Calculation of Akaike information criterion and Bayesian Information Criteria

Akaike is a statistical method that is used to determine the model which truly explains the variance in the dependent variable with the fewest number of independent variables.

Calculation of AIC for each model

Using the maximum likelihood estimate, the relative information value of the model and the number of parameters in the model are determined. The formula for AIC is expressed as:

$$\mathrm{AIC} = 2k - 2\ln\widehat{L}$$

Where:

- k :- number of parameters in the model
- L I :- the maximum likelihood of the estimated model

The results obtainedare listed below:

Model	**AIC**
Model1	233.3414
Model2	-334.6489
Model3	744.2702
Model4	117.1799
Model5	-333.0493

Calculation of BIC for each model

A criterion for model selection among a finite set of models is called Bayesian information criterion. It is possible to increase the likelihood by adding parameters while fitting the models. It is based, in part, on the likelihood function, and it is closely related to Akaike information criterion (AIC). We can resolve this problem using BIC by introducing a penalty term for the number of parameters in the model. AIC.BIC has been widely used for model identification in time series and linear regression and been so applied quite widely to any set of maximum likelihood-based models.

Mathematically BIC can be defined as:

$$\mathrm{BIC} = \ln(n)k - 2\ln(\hat{L}).$$

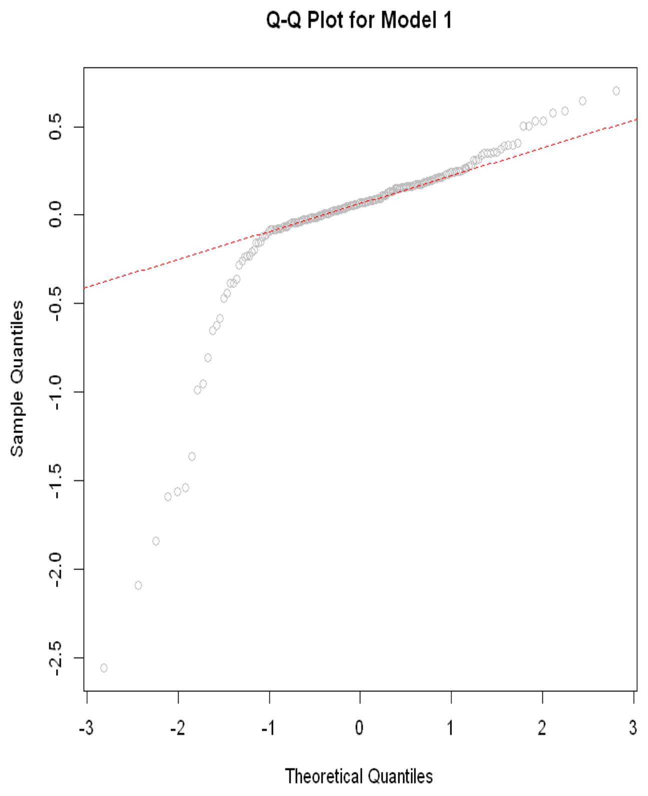
k is the number of free parameters to be estimated

n is the number of data points

$\hat{L}$ is the maximized value of the likelihood function of the model

The results obtained are listed below:

Model	BIC
Model1	253.1613
Model2	-321.4357
Model3	754.1801
Model4	130.3931
Model5	-316.5328



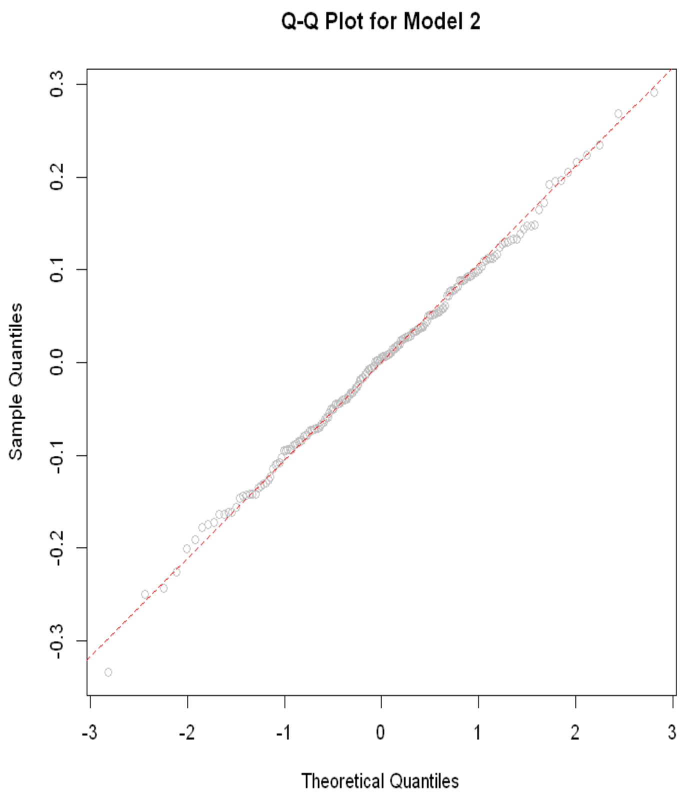
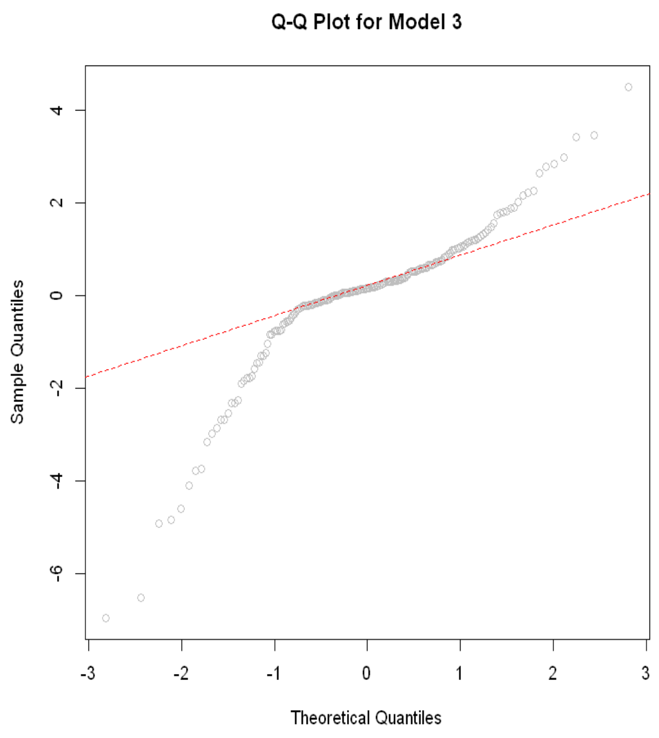
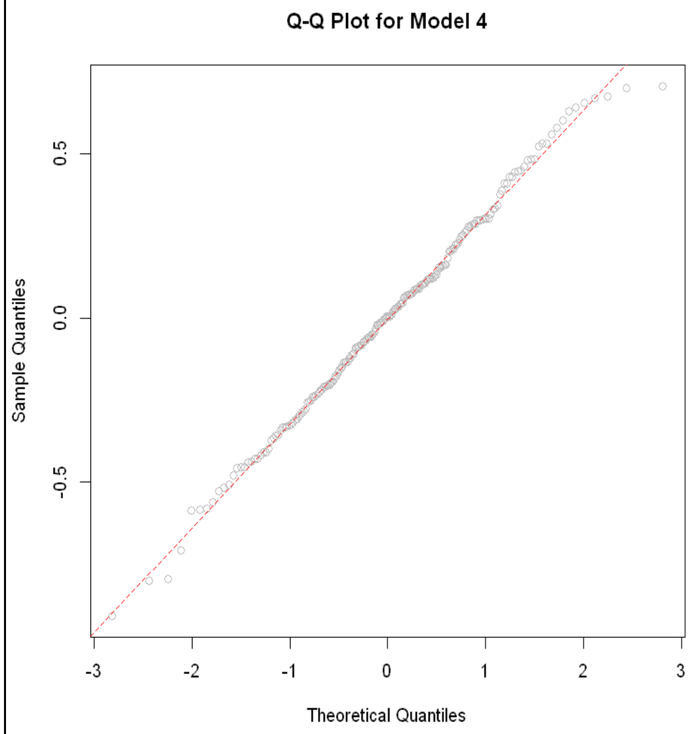


Fig13: Figure of Model 1 and Model 2 Q-Q plot





-

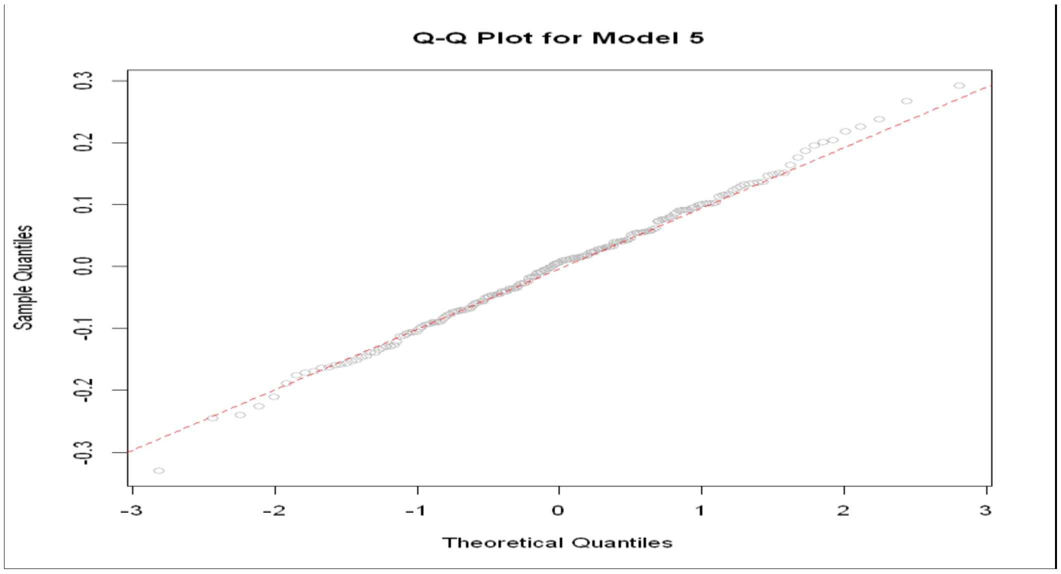


Fig 14: Figure of Model 3, Model4 and Model 5 Q-Q plot

Selecting the best regression model

Now, based on BIC and AIC best model would be model 2. It has lowest value. We will observe the Q-Q plot. Model 2 and model 5 seems to a have normal distribution. After observing and calculating the distribution of each mode, Now let us see if model 2 is more suitable the than others or not. Looking the each length of the parameter, the lowest is model 3 with 3 numbers of parameter, but it doesn't follows a normal distribution and it is skewed and the next is model 4 having the number of 4 parameters but its AIC and BIC is greater than the mod el 2.So in conclusion of AIC, BIC, Q-Q plot and extra interpretability, we have picked up model 2 as the best-fit model and which is expressed as:
Model 2: $y = \theta 1\ x4 + \theta 2\ x13 + \theta 3\ x34 + \theta bias + \varepsilon$

Splitting training and testing data for selected model 2

We found the best model as Model 2. Each dataset was divided according to the information provided (70% for training and 30% for testing. With the help of the selected best model estimation of the model, parameters were carried out on training data.

We are using a 70/30 ratio because it provides a good balance between having enough data (70%) for the model to learn from during training and having quite reasonable (30%) data to test. So, using initial_split () function, we will separate data in 70/30 by providing the numbers into the prop argument of the function for both y and x signals.

Computation of 95% confidence interval and visualization

The calculated t-value is -2390.2. The sample mean of y_training_data is not equal but very far from the hypothesized mean of 500 following a negative direction (minus sign).

We can see that p-value is less than 2.2e (is much less than 0.05). So there is strong evidence against the null hypothesis and in favor of the alternative hypothesis (that the true mean is not equal to 500). From observation, At 95% confidence level, we found that y_training_data is significantly different from 500, So, it indicates that the null hypothesis is not likely to be true. So finally we can say that the mean of y_training_data is equal to 500 (null hypothesis) is not likely to be true.

The lowest and upper limit values are -0.1455669 and 0.6908340. There is a 95% chance that the true mean of y_training_data falls within this interval.

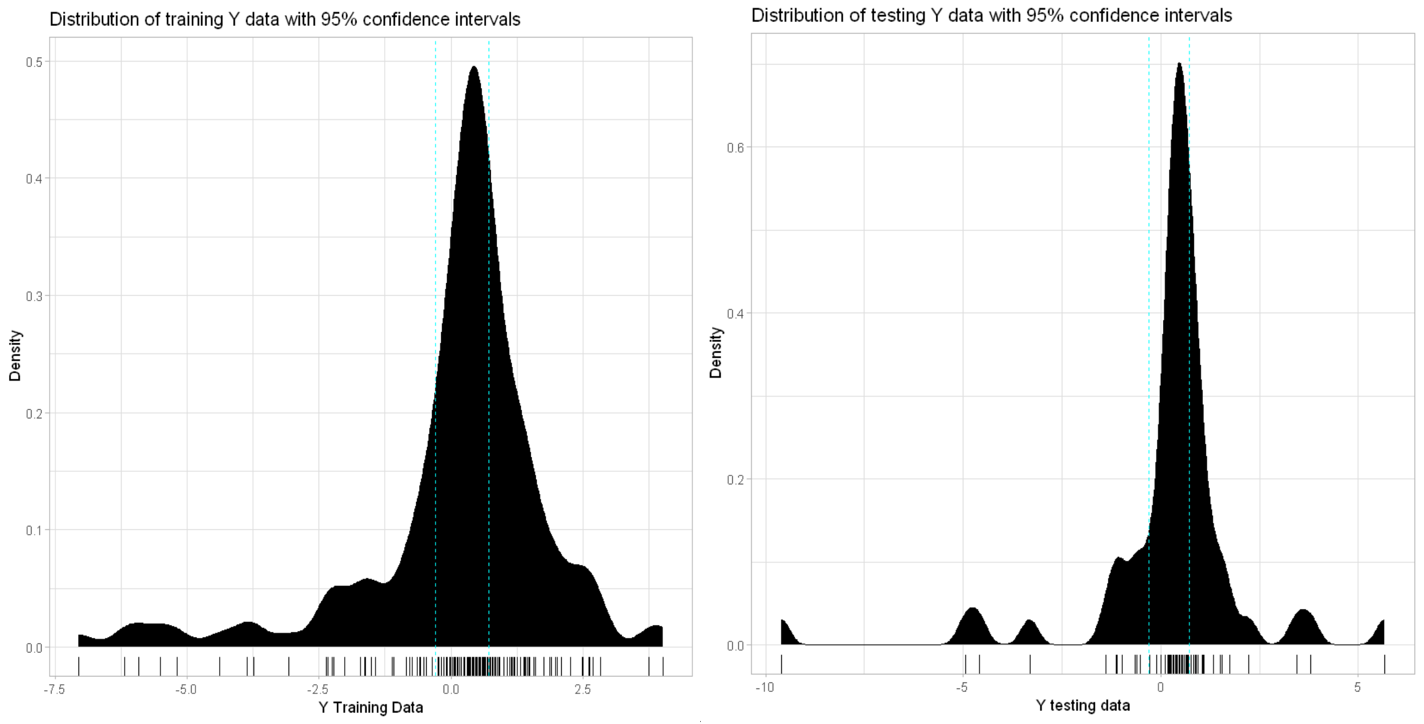


Fig 15: Distribution of training and testing data with 95% confidence intervals

3. Approximate Bayesian Computation (ABC)

Bayesian statistics are methods that allow for the systematic updating of beliefs in the evidence of new data. Bayes' theorem describes the probability of occurrence of an event related to any condition. It is also considered for the case of conditional probability. One of the many applications of Bayes' theorem is Bayesian inference, a particular approach to statistical inference. In Probability, Bayes theorem is a mathematical formula, which is used to determine the conditional probability of the given event.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where P(A|B) represents the probability of A given B has happened and P(B|A) represents the probability of B given A has happened, P(A) and P(B) are the probabilities of A and B happening regardless of anything else respectively.

Now again, model 2 is selected as a best regression module. Four values were selected from model 2 for estimating model parameters which were carried out on task 2.1. The selected parameter must be the highest possible value. Using the least squares of model 2 theta bias and theta one was selected for calculation of the posterior value. Two values of estimated parameters are kept constant.

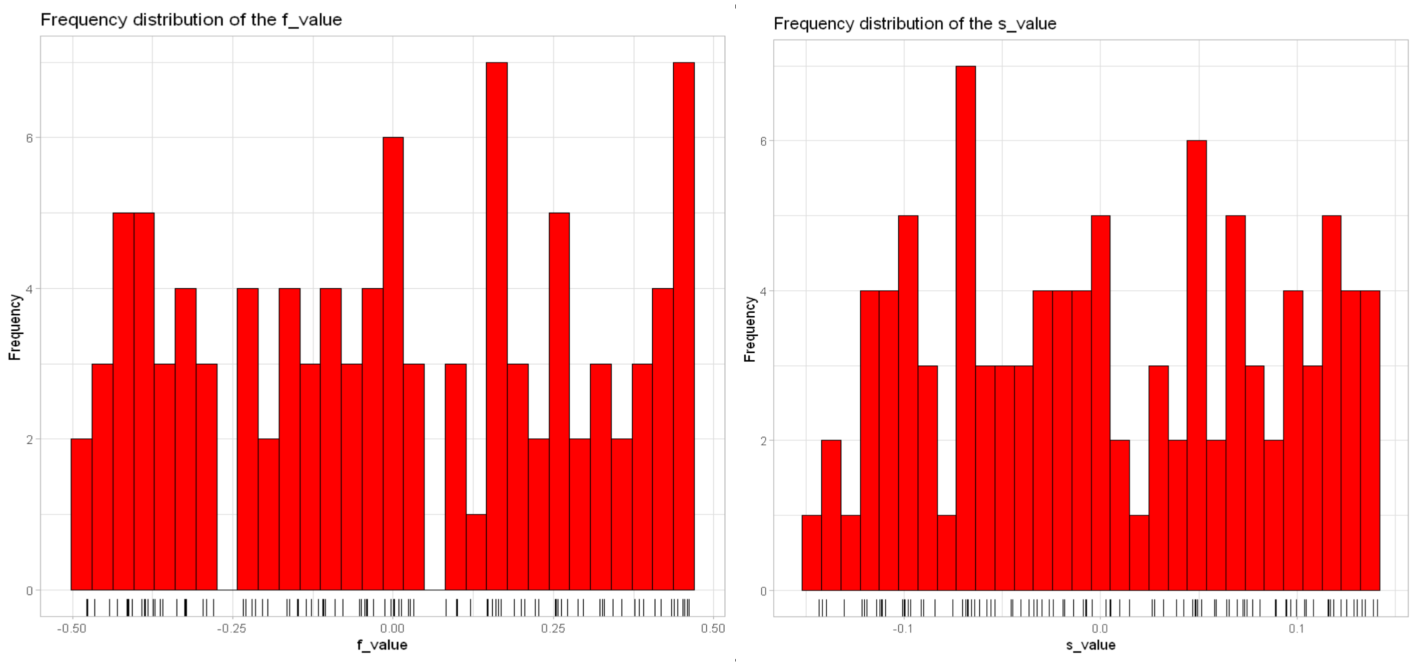


Fig16: Figure of Frequency distribution of the f_value and s_value

Visualization of Joint and marginal posterior distribution for the parameters plot

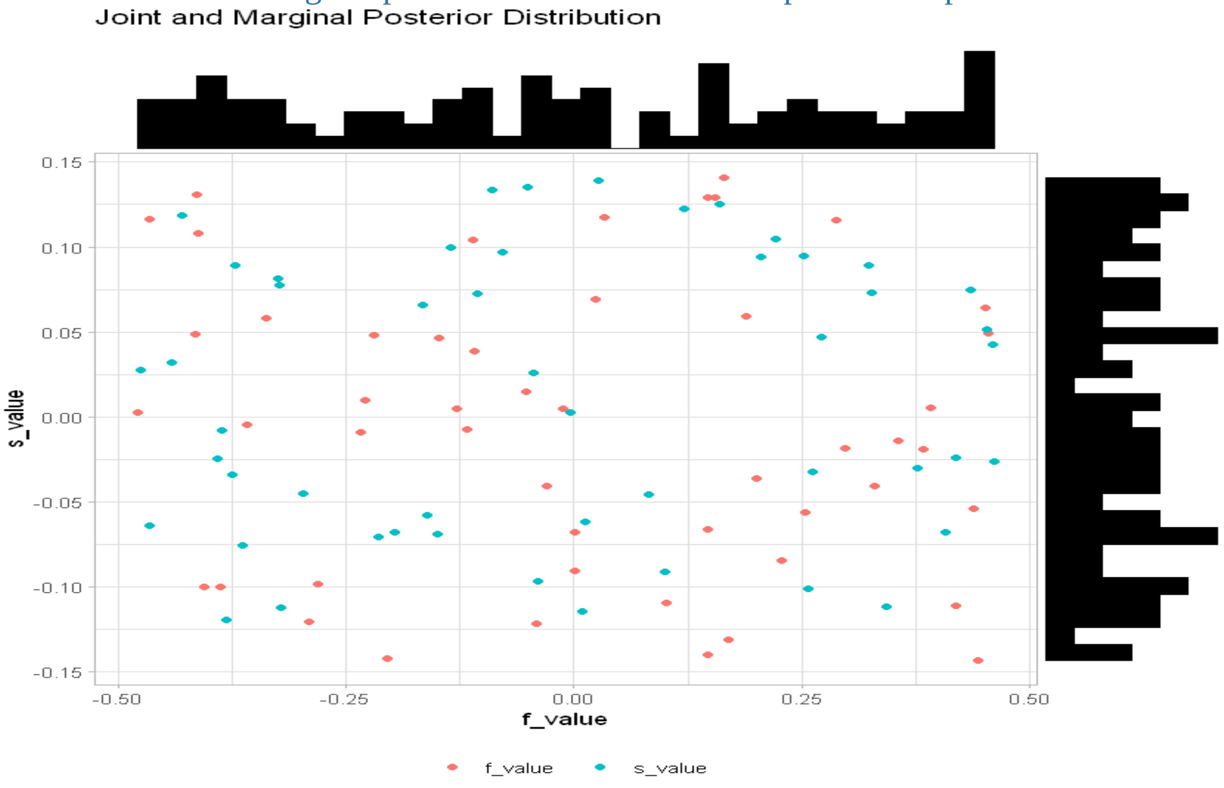


Fig17: Figure of Joint and Marginal Posterior Distribution

Conclusion

Now finally, our objectives have been achieved to a satisfactory degree. I selected the best regression model from all those candidate models. **Model 2 was selected as the best regression model** while carrying out each task. But, I found something that should be improved. A sufficient amount of data is required for Polynomial regression to increase accuracy. The model might be overfitting even if the model is trained too well. Non-linear, ECG signals is not suitable for polynomial regression.

2-appendix

February 9, 2023

1 Appendix

Aayush Adhikari 220426@softwarica.edu.np

1.0.1 https://github.com/kripaz777/modeling_eeg

[ ]:
```
library(ggplot2)
library(GGally)
library(tidyverse)
library(tidymodels)
library(moments)
library(dplyr)
library(ggExtra)
library(readr)
library(ggpubr)
```

1.0.2 Importing required library

[ ]:
```
library(ggplot2) # implementation of the grammar of graphics - complex␣
 ↪visualizations
library(GGally)  # create a matrix of scatter plots, with histograms or density␣
 ↪plots
library(tidyverse) # data manipulation, exploration, and visualization
library(tidymodels)  # Split the data into training and testing
library(moments) # calculating and plotting descriptive statistics
library(dplyr) # data manipulation toolkit for working with structured data
library(ggExtra) # additional functionality to ggplot2
library(readr) # fast and friendly file reading
library(ggpubr) # customizing ggplot2 based publication ready plots

#setting the theme to light
theme_set(theme_light())
```

1.0.3 Importing data set

[ ]:
```
# Importing X data
x=as.matrix(read.csv(file="X_1673241366257.csv",header = F))
```

[ ]:
```
# Importing Y data
y=as.matrix(read.csv(file="y_1673241374123.csv",header = F))
```

[ ]:
```
time = read.csv("time_1673241270748.csv", header = F, skip = 1)
time = as.matrix(rbind(0, time))
```

[ ]:
```
colnames(x) <- paste0(rep("x", ncol(x)), 1:ncol(x))
colnames(y) <- "y"
colnames(time) <- "time"
df = cbind(time, x, y)
```

[ ]:
```
colnames(x)<-c("x1","x2","x3","x4")
xmat <- as.matrix(x)
```

[ ]:
```
#Similar process as above for output data Y.

ymat <- as.matrix(y)
```

[ ]:
```
#Similar process as above for output data Y.

time_mat <- as.matrix(time)
```

1.1 1: Preliminary data analysis

1.1.1 Plots of Input and Output Plot od EEG Time Series Signals

[ ]:
```
time_mat <- as.matrix(time)
```

[ ]:
```
# Using is.na function to see
missing_values <- is.na(df)
missvalues <- sum(missing_values)

print(sprintf("No of missing value in the dataset is %d", missvalues))
```

[ ]:
```
xmat.ts<-ts(xmat,start = c(min(time_mat),max(time_mat)),frequency =1)
plot(xmat.ts, xlab = "Time", ylab = "Input signal",main = "Time series plot of␣
 ↪Input Signals")
```

[ ]:
```
ymat.ts<-ts(ymat,start = c(min(time),max(time)),frequency =1)
plot(ymat.ts, xlab = "Time", ylab = "Output signals",main = "Time series plot␣
 ↪of output Signals")
```

1.1.2 Distributions for each EEG signals

[ ]:
```
# Creating a Histrogram of X signal
dis=density(x)
plot(dis,main = "Figure of Density plot of input signals")

hist(x,freq = FALSE,main = "DEn")
lines(dis,lwd=2,col="blue")
rug(jitter(x))
```

[ ]:
```
```

[ ]:
```
#Density plot of X1 signal
density_X1=density(x[,"x1"])
hist(x[,"x1"],main = "Figure of Histogram and density plot of X1",freq =␣
  ↪FALSE,xlab = "X1 Signal")
lines(density_X1,lwd=2,col="red")
# Add the data-poins with noise in the X-axis
rug(jitter(x[,"x1"]))
```

[ ]:
```
#Density plot of X2 signal
density_X2=density(x[,"x2"])
hist(x[,"x2"],main = "Figure of Histogram and density plot of X2",freq =␣
  ↪FALSE,xlab = "X2 Signal")
lines(density_X2,lwd=2,col="red")
rug(jitter(x[,"x2"]))
```

[ ]:
```
#Density plot of X3 signal
density_X3=density(x[,"x3"])
hist(x[,"x3"],main = "Figure of Histogram and density plot of X3",freq =␣
  ↪FALSE,xlab = "X3 Signal")
lines(density_X3,lwd=2,col="red")
rug(jitter(x[,"x3"]))
```

[ ]:
```
#Density plot of X4 signal
density_X4=density(x[,"x4"])
hist(x[,"x4"],main = "Figure of Histogram and Density plot of X4",freq =␣
  ↪FALSE,xlab = "X4 Signal")
lines(density_X4,lwd=2,col="red")
rug(jitter(x[,"x4"]))
```

[ ]:
```
#Creating a density of Y signal
density_y=density(y)
plot(density_y,main = "Figure of Density plot of Y",xlab = "Output Signal")
hist(y,main = "Figure of Histogram and density plot of Y", freq = FALSE,xlab =␣
  ↪"Output Signal")
lines(density_y,lwd=2,col="red")
```

```
rug(jitter(y))
```

1.1.3 Correlation and scatter plots (between different input EEG signals and the output EEG) to examine their dependencies

```
[ ]: # Plot of X1 with Y
     plot(x[,"x1"],y, main = "Plot of Correlation betweeen X1 and Y signal", xlab =␣
      ↪"X1 signal", ylab = "Output signal" )

     # Plot of X2 with Y
     plot(x[,"x2"],y, main = "Plot of Correlation betweeen X2 and Y signal",xlab =␣
      ↪"X2 signal", ylab = "Output signal")
     # Plot of X3 with Y
     plot(x[,"x3"],y, main = "Plot of Correlation betweeen X3 and Y signal", xlab =␣
      ↪"X3 signal", ylab = "Outputsignal")
     # Plot of X4 with Y
     plot(x[,"x4"],y, main = "Plot of Correlation betweeen X4 and Y signal", xlab =␣
      ↪"X4 signal", ylab = "Outputsignal")
```

1.2 Task 2: Regression – modeling the relationship between EEG signals

1.2.1 Using least square, estimating model parameter

```
[ ]: ones_value = matrix(1 , length(x)/4,1)
     #The resulting data frame "x_model_1" will have the same number of rows as the␣
      ↪input variable "x" but new colum will be added,

     x_value_model_1<-cbind(ones_value,(x[,"x4"]),(x[,"x1"])^2,(x[,"x1"])^3,(x[,"x2"])^4,(x[,"x1"])
```

```
[ ]: # We are converting thus obtained dataframe x_model_1  into a matrix using the␣
      ↪function as.matrix(). The transpose of the matrix is taken using the␣
      ↪function t(). The operator %*% is used for matrix multiplication under␣
      ↪solve() function.

     theta_hat_1 <- solve(t(as.matrix(x_value_model_1)) %*% as.
      ↪matrix(x_value_model_1)) %*% t(as.matrix(x_value_model_1)) %*% as.matrix(y)
```

1.2.2 Using least square estimating model parameter for model 2

```
[ ]: # Binding for model 2
     x_value_model_2<-cbind(ones_value,(x[,"x4"]),(x[,"x1"])^3,(x[,"x3"])^4)

     # Calculation of theta_hat_2
     theta_hat_2 <- solve(t(as.matrix(x_value_model_2)) %*% as.
      ↪matrix(x_value_model_2)) %*% t(as.matrix(x_value_model_2)) %*% as.matrix(y)
```

1.2.3 Using least square, estimating model parameter for model 3

[ ]:
```
# Binding for model 3
x_value_model_3<-cbind(ones_value,(x[,"x3"])^3,(x[,"x3"])^4)

# Calculation of theta_hat
theta_hat_3 <- solve(t(as.matrix(x_value_model_3)) %*% as.
↪matrix(x_value_model_3)) %*% t(as.matrix(x_value_model_3)) %*% as.matrix(y)
```

1.2.4 Using least square, estimating model parameter for model 4

[ ]:
```
# Binding for model 4
x_value_model_4<-cbind(ones_value,(x[,"x2"]),(x[,"x1"])^3,(x[,"x3"])^4)

# Calculation of theta_hat
theta_hat_4 <- solve(t(as.matrix(x_value_model_4)) %*% as.
↪matrix(x_value_model_4)) %*% t(as.matrix(x_value_model_4)) %*% as.matrix(y)
```

1.2.5 Using least square, estimating model parameter for model 5

[ ]:
```
# Binding for model 5
x_value_model_5 <-␣
↪cbind(ones_value,(x[,"x4"]),(x[,"x1"])^2,(x[,"x1"])^3,(x[,"x3"])^4)

# Calculation of theta_hat
theta_hat_5 <- solve(t(as.matrix(x_value_model_5)) %*% as.
↪matrix(x_value_model_5)) %*% t(as.matrix(x_value_model_5)) %*% as.matrix(y)
```

1.2.6 Calculation of model residual errors (RSS)

Calculation of model residual errors (RSS) for model 1

[ ]:
```
# Before that we convert our model and theta hat into matrix
x_model_1 <- as.matrix(x_model_1)
theta_hat_1 <- as.matrix(theta_hat_1)


Y_hat_model_1 <- x_model_1 %*% theta_hat_1



# Calculation of RSS using above mentioned formula

RSS_value_model_1 <- sum((y - Y_hat_model_1)^2)

# printing RSS value for model 1
print(sprintf("RSS value of the model 1 =  %0.4f", RSS_value_model_1))
```

1.2.7 Calculation of model residual errors (RSS) for model 2

[ ]:
```
# Before that we convert our model and theta hat into matrix
x_model_2 <- as.matrix(x_model_2)
theta_hat_2 <- as.matrix(theta_hat_2)


Y_hat_model_2 <- x_model_2 %*% theta_hat_2




# Calculation of RSS using above mentioned formula

RSS_value_model_2 <- sum((y - Y_hat_model_2)^2)

# printing RSS value for model 2

print(sprintf("RSS value of the model 2 =  %0.4f", RSS_value_model_2))
```

1.2.8 Calculation of model residual errors (RSS) for model 3

[ ]:
```
# Before that we convert our model and theta hat into matrix
x_model_3 <- as.matrix(x_model_3)
theta_hat_3 <- as.matrix(theta_hat_3)


Y_hat_model_3 <- x_model_3 %*% theta_hat_3




# Calculation of RSS

RSS_value_model_3 <- sum((y - Y_hat_model_3)^2)

# printing RSS value for model 3

print(sprintf("RSS value of the model 3 =  %0.4f", RSS_value_model_3))
```

1.2.9 Calculation of model residual errors (RSS) for model 4

[ ]:
```
# Before that we convert our model and theta hat into matrix
x_model_4 <- as.matrix(x_model_4)
theta_hat_4 <-  as.matrix(theta_hat_4)


Y_hat_model_4 <- x_model_4 %*% theta_hat_4
```

```
# Calculation of RSS

RSS_value_model_4 <- sum((y - Y_hat_model_4)^2)

# printing RSS value for model 4

print(sprintf("RSS value of the model =  is %0.4f", RSS_value_model_4))
```

1.2.10 Calculation of model residual errors (RSS) for model 5

```
[ ]: # Calculation of Y-hat and RSS model 5

# Before that we convert our model and theta hat into matrix
x_model_5 <- as.matrix(x_model_5)
theta_hat_5 <- as.matrix(theta_hat_5)


Y_hat_model_5 <- x_model_5 %*% theta_hat_5



# Calculation of RSS using

RSS_value_model_5 <- sum((y - Y_hat_model_5)^2)

# printing RSS value for model 5

print(sprintf("RSS value of the model 5 =  is %0.4f", RSS_value_model_5))
```

1.2.11 Calculation of log-likelihood functions

```
[ ]: # Calculation of length of the output signal y with nrow()  as our data are in␣
 ↪matrix format and storing it in N

N <- nrow(y)


# Calculation of the variance of model 1

Variance_value_model_1 = RSS_value_model_1/(N-1)

# Printing variance of model 1

print(sprintf("Calculated variance of model 1 =  %0.4f",␣
 ↪Variance_value_model_1))
```

```
[ ]: # Calculation of the log-likelihood of model 1 using model residual error (RSS)

likehood_value_model_1 <- -(N/2)*(log(2*pi))-(N/
 ↪2)*(log(Variance_value_model_1))-(1/
 ↪(2*Variance_value_model_1))*RSS_value_model_1

# Printing log likelihood function of model 1
print(sprintf("Calculated Log-likelihood of model 1 =  %0.4f",␣
 ↪likehood_value_model_1))
```

Calculation of log-likelihood functions for model 2

```
[ ]: # Calculation of the variance of model 2

Variance_value_model_2 = RSS_value_model_2/(N-1)

# Printing variance of model 2

print(sprintf("Calculated variance of model 2 = %0.4f", Variance_value_model_2))

# Calculation of the log-likelihood of model 2 using model residual error (RSS)

likehood_value_model_2 <- -(N/2)*(log(2*pi))-(N/
 ↪2)*(log(Variance_value_model_2))-(1/
 ↪(2*Variance_value_model_2))*RSS_value_model_2

# Printing log likelihood function of model 2
print(sprintf("Calculated Log-likelihood of model 2 =  %0.4f",␣
 ↪likehood_value_model_2))
```

Calculation of log-likelihood functions for model 3

```
[ ]: # Calculation of the variance of model 3

Variance_value_model_3 = RSS_value_model_3/(N-1)

# Printing variance of model 3

print(sprintf("Calculated variance of model 3 is %0.4f",␣
 ↪Variance_value_model_3))


# Calculation of the log-likelihood of model 3 using model residual error (RSS)

likehood_value_model_3 <- -(N/2)*(log(2*pi))-(N/
 ↪2)*(log(Variance_value_model_3))-(1/
 ↪(2*Variance_value_model_3))*RSS_value_model_3
```

```
# Printing log likelihood function of model 3
print(sprintf("Calculated Log-likelihood of model 3 =  %0.4f",␣
↪likehood_value_model_3))
```

Calculation of log-likelihood functions for model 4

```
[ ]: Variance_value_model_4 = RSS_value_model_4/(N-1)

# Printing variance of model 4

print(sprintf("Calculated variance of model 4 is %0.4f",␣
↪Variance_value_model_4))



# Calculation of the log-likelihood of model 4 using model residual error (RSS)

likehood_value_model_4 <- -(N/2)*(log(2*pi))-(N/
↪2)*(log(Variance_value_model_4))-(1/
↪(2*Variance_value_model_4))*RSS_value_model_4

# Printing log likelihood function of model 4
print(sprintf("Calculated Log-likelihood of model 4 = %0.4f",␣
↪likehood_value_model_4))
```

Calculation of log-likelihood functions for model 5

```
[ ]: # Calculation of the variance of model 5

Variance_value_model_5 = RSS_value_model_5/(N-1)

# Printing variance of model 5

print(sprintf("Calculated variance of model 5 = %0.4f", Variance_value_model_5))



# Calculation of the log-likelihood of model 5 using model residual error (RSS)

likehood_value_model_5 <- -(N/2)*(log(2*pi))-(N/
↪2)*(log(Variance_value_model_5))-(1/
↪(2*Variance_value_model_5))*RSS_value_model_5

# Printing log likelihood function of model 5
print(sprintf("Calculated Log-likelihood of model 5 = %0.4f",␣
↪likehood_value_model_5))
```

1.2.12 Calculation of Akaike information criterion and Bayesian Information Criteria

Calculation of AIC for model 1

```
[ ]: # Calculation of AIC for model 1.
     # Here we are finding value of K with length() function.

     AIC_1 <- 2* length(x_model_1[1,]) - 2 * likehood_value_model_1

     print(sprintf("Length of parameter to be estimated in model 1 is %d",␣
      ↪length(x_model_1[1,])))

     # AIC of model 1
     print(sprintf("Calculated AIC of model 1 is %0.4f", AIC_1))

     # Calculation of AIC for model 2.


     AIC_2 <- 2* length(x_model_2[1,]) - 2 * likehood_value_model_2

     print(sprintf("Length of parameter to be estimated in model = is %d",␣
      ↪length(x_model_2[1,])))

     # AIC of model 2
     print(sprintf("Calculated AIC of model 2 = %0.4f", AIC_2))
```

Calculation of AIC for model 2

```
[ ]: # Calculation of AIC for model 2.


     AIC_2 <- 2* length(x_model_2[1,]) - 2 * likehood_value_model_2

     print(sprintf("Length of parameter to be estimated in model 2 = %d",␣
      ↪length(x_model_2[1,])))

     # AIC of model 2
     print(sprintf("Calculated AIC of model 2 = %0.4f", AIC_2))
```

Calculation of AIC for model 3

```
[ ]: # Calculation of AIC for model 3.


     AIC_3 <- 2* length(x_model_3[1,]) - 2 * likehood_value_model_3

     print(sprintf("Length of parameter to be estimated in model 3 = %d",␣
      ↪length(x_model_3[1,])))

     # AIC of model 3
     print(sprintf("Calculated AIC of model 3 = %0.4f", AIC_3))
```

Calculation of AIC for model 4

[ ]:
```
# Calculation of AIC for model 4.


AIC_4 <- 2* length(x_model_4[1,]) - 2 * likehood_value_model_4

print(sprintf("Length of parameter to be estimated in model 4 = %d",␣
 ↪length(x_model_4[1,])))

# AIC of model 4
print(sprintf("Calculated AIC of model 4 = %0.4f", AIC_4))
```

Calculation of AIC for model 5

[ ]:
```
# Calculation of AIC for model 5.


AIC_5 <- 2* length(x_model_5[1,]) - 2 * likehood_value_model_5

print(sprintf("Length of parameter to be estimated in model 5 = %d",␣
 ↪length(x_model_5[1,])))

# AIC of model 5
print(sprintf("Calculated AIC of model 5 = %0.4f", AIC_5))
```

1.2.13 Calculation of BIC for each models

Calculation of BIC for model 1

[ ]:
```
BIC_value_1 <- length(x_model_1[1,]) * log(N) - 2 * likehood_value_model_1


# BIC of model 1
print(sprintf("Calculated BIC of model 1 =  %0.4f", BIC_value_1))
```

Calculation of BIC for model 2

[ ]:
```
# Calculation of BIC for model 2

BIC_value_2 <- length(x_model_2[1,]) * log(N) - 2 * likehood_value_model_2


# BIC of model 2
print(sprintf("Calculated BIC of model 2 = %0.4f", BIC_value_2))
```

Calculation of BIC for model 3

[ ]:
```
BIC_value_3 <- length(x_model_3[1,]) * log(N) - 2 * likehood_value_model_3
```

```
# BIC of model 3

print(sprintf("Calculated BIC of model 3 = %0.4f", BIC_value_3))
```

Calculation of BIC for model 4

```
[ ]: # Calculation of BIC for model 4

     BIC_value_4 <- length(x_model_4[1,]) * log(N) - 2 * likehood_value_model_4


     # BIC of model 4

     print(sprintf("Calculated BIC of model 4 = %0.4f", BIC_value_4))
```

Calculation of BIC for model 5

```
[ ]: BIC_value_5 <- length(x_model_5[1,]) * log(N) - 2 * likehood_value_model_5


     # BIC of model 5

     print(sprintf("Calculated BIC of model 5 = %0.4f", BIC_value_5))
```

1.3 Checking distribution of model prediction errors

```
[ ]: ##  Error of models 1-5  based on calculation form Task 2.2
     model_1_error <- y - Y_hat_model_1
     model_2_error <- y - Y_hat_model_2
     model_3_error <- y - Y_hat_model_3
     model_4_error <- y - Y_hat_model_4
     model_5_error <- y - Y_hat_model_5
```

1.3.1 Visualization of prediction error with Q-Q plot

```
[ ]: # For model 1

     qqnorm(t(model_1_error),col = "grey", main = "Q-Q plots of prediction error for␣
      ↪model 1" )
     qqline(model_1_error, col = "green", lwd = 1,lty = 2)
```

```
[ ]: # For model 2

     qqnorm(t(model_2_error), main = "Q-Q plots of prediction error for model␣
      ↪2",col= "grey",  )
     qqline(model_2_error, col = "green", lwd = 1,lty = 2)
```

```
[ ]: #For model 2

     qqnorm(t(model_2_error),main = "Q-Q plots of prediction error for model 2",col=␣
      ↪"grey")
     qqline(model_2_error, col = "green", lwd = 1,lty = 2)
```

```
[ ]: #  For model 3

     qqnorm(t(model_3_error), main = "Q-Q plots of prediction error for model 3",␣
      ↪col= "grey" )
     qqline(model_3_error, col = "green", lwd = 1,lty = 2)
```

```
[ ]: # For model 4

     qqnorm(t(model_4_error), main = "Q-Q plots of prediction error for model␣
      ↪4",col= "grey" )
     qqline(model_4_error, col = "green", lwd = 1, lty = 2)
```

```
[ ]: # For model 5

     qqnorm(t(model_5_error), main = "Q-Q plots of prediction error for model␣
      ↪5",col= "grey" )
     qqline(model_5_error, col = "green", lwd = 1,lty = 2)
```

1.3.2 Selecting best model

```
[ ]: par(mfrow = c(3,2))
     hist (model_1_error[,1], freq = FALSE, col="blue", las =1)
     abline(v = median(model_1_error[,1]), col = "grey", lwd = 5)
     abline(v = mean(model_1_error[,1]), col = "purple", lwd = 5)


     hist (model_2_error[,1], freq = FALSE, col="green", las =1)
     abline(v = median(model_2_error[,1]), col = "grey", lwd = 5)
     abline(v = mean(model_2_error[,1]), col = "purple", lwd = 5)
     # abline(v = getmode(model_2_error[,1]), col = "red", lwd = 5)

     hist (model_3_error[,1], freq = FALSE, col="orange", las =1)
     abline(v = median(model_3_error[,1]), col = "grey", lwd = 5)
     abline(v = mean(model_3_error[,1]), col = "purple", lwd = 5)


     hist (model_4_error[,1], freq = FALSE, col="yellow", las =1)
     abline(v = median(model_4_error[,1]), col = "grey", lwd = 5)
     abline(v = mean(model_4_error[,1]), col = "purple", lwd = 5)
```

```
hist (model_5_error[,1], freq = FALSE, col="pink", las =1)
abline(v = median(model_5_error[,1]), col = "grey", lwd = 5)
abline(v = mean(model_5_error[,1]), col = "purple", lwd = 5)


hist(0, main = "color code")
legend("center", legend = c("median","mean"),
       lwd = 1, col = c("grey", "purple"))
```

1.4 Splitting data for test and train for selected model 2

[ ]:
```
# Splitting input signals x

split_data_x <- initial_split(data = as.data.frame(x),prop=.7)

x_training_data_set <- training(split_data_x)

x_training_data <- as.matrix(x_training_data_set)

x_testing_data_set <- testing(split_data_x)

x_testing_data <- as.matrix(x_testing_data_set)


# Splitting the data

split_data_y <- initial_split(data = as.data.frame(y),prop=.7)



# Training  data for output signals y  are split and thus splitted training set␣
 ↪is converted to matrix form ans assigned to the variable y_training_data

y_training_data_set <- training(split_data_y)

y_training_data <- as.matrix(y_training_data_set)

# Testing data for output signals y are splitted and thus splitted testing set␣
 ↪is converted to matrix form ans assigned to the variable y_testing_data

y_testing_data_set <- testing(split_data_y)

y_testing_data <- as.matrix(y_testing_data_set)
```

1.5 Estimation of model parameter for selected model 2 using training dataset

[ ]:
```
#   Estimation of model parameters using training set

training_data_ones <- matrix(1, length(x_testing_data_set[["x1"]]),1)


# cbind() is used to bind the matrix "ones" and other variables that are␣
 ↪present in that model 1, together to create a new data frame, "x_model_1".

#The resulting data frame "x_model_1" will have the same number of rows as the␣
 ↪input variable "x" but new colum will be added,

training_model_x <-␣
 ↪cbind(training_data_ones,(x_testing_data_set[["x4"]]),(x_testing_data_set[["x1"]])^3,(x_tes


training_thetahat <-   solve(t(training_model_x) %*% training_model_x) %*%␣
 ↪t(training_model_x) %*% y_testing_data
```

1.6 Model output prediction on testing data set

[ ]:
```
#  y_testing_hat is being calculated as the product of the x_testing_data␣
 ↪matrix and the estimated coefficients (training_thetahat) of the linear␣
 ↪regression model.
# Here we get predicted values of y for the testing data.

y_testing_hat <- x_testing_data %*% training_thetahat


# Calculating residual sum of squares (RSS) for the testing data
RSS_testing <- sum((y_testing_data_set-y_testing_hat)^2)


# Printing RSS value of testing

print(sprintf("RSS value is testing data %0.4f", RSS_testing))
```

1.7 Computation of 95% confidence interval and visualizaing

[ ]:
```
# Performing t-test on the training data. Here, mean is taken as a null␣
 ↪hypothesis which value is equal to 500,
# "two.sided"  is used which says that the true mean is not equal to the␣
 ↪hypothesized mean and setting confidence interval 95%

t.test(y_testing_data, mu=500, alternative="two.sided", conf.level=0.95)
```

```
[ ]: # Setting CI value based on t-test at the time of run
     C_I1 <- -0.2974446

     C_I2 <- 0.7263838


     ggplot(data = data.frame(y_training_data), aes(x = y_training_data)) +
       geom_density(col = "black", fill = "black" , lw=1) +
       geom_vline(xintercept = C_I1, col = "cyan", linetype = "dashed") +
       geom_vline(xintercept = C_I2, col = "cyan", linetype = "dashed") +
       geom_rug()+

       ggtitle("Distribution of training Y data with 95% confidence intervals")+
       xlab("Y Training Data") +
       ylab("Density")
```

```
[ ]: # Plotting distribution of testing data


     ggplot(data = data.frame(y_testing_data), aes(x = y_testing_data)) +
       geom_density(col = "black", fill = "black" , lw=1) +
       geom_vline(xintercept = C_I1, col = "cyan", linetype = "dashed") +
       geom_vline(xintercept = C_I2, col = "cyan", linetype = "dashed") +
       geom_rug()+

       ggtitle("Distribution of testing Y data with 95% confidence intervals")+
       xlab("Y testing data") +
       ylab("Density")
```

```
[ ]: # Error between actual testing data and the model's predicted output on that␣
      ↪data

     error <- ((y_testing_data_set - y_testing_hat))



     # Printing error value

     print(sprintf("Error between actual testing data and the model's predicted␣
      ↪output on that data is  %0.4f", RSS_testing))
```

1.8 Task 3: Approximate Bayesian Computation (ABC)

1.9 Computation of 2 parameter posterior distributions

[ ]:
```
# Creator vector of theta_hat of selected model 2 and sorting them to find out␣
↪two largest absolute value and printing it

numbers <- c(theta_hat_2)
sorted_numbers <- sort(abs(numbers), decreasing=TRUE)
largest_two_values <- sorted_numbers[1:2]
print(largest_two_values)
```

[ ]:
```
#Choosing parameters

theta_bias <- largest_two_values[1]
theta_four <- largest_two_values[2]

#Constant parameter

theta_one  <-  0.010038614
theta_three   <- -0.001912836


# Initial values

arr_1 = 0
arr_2=0
f_value=0
s_value=0
```

1.9.1 Using Uniform distribution as prior, around the estimated parameter values for 2 parameters

[ ]:
```
# Calculating epsilon

epsilon <- RSS_value_model_2 * 2

num <- 100

##Calculating Y-hat

counter <- 0

# Iteration from 0 -100 and calculating the range

for (i in 1:num) {
r1 <- runif(1,-0.483065688,0.483065688)
r2 <- runif(1,-0.1435789,0.1435789)

# Let us create a new vector of  two values from range1 and range2
thetahat1 <- matrix(c(r1,r2,theta_one,theta_three))

# Calculating predicted response values for the current iteration of the loop

New_Y_Hat <- x_model_2 %*% thetahat1

# Calculting new RSS valur
RSS_new <- sum((y - New_Y_Hat)^2)



# Checking which is greater new RSS or epsilon

if (RSS_new > epsilon){
arr_1[i] <- r1
arr_2[i] <- r2
counter = counter+1
f_value <- matrix(arr_1)
s_value <- matrix(arr_2)

  }
}
```

1.9.2 Plotting the joint and marginal posterior distribution for the parameters

[ ]:
```
# Plotting histogram of new f_values and s_values

# Frequency distribution of the f_value
ggplot(data.frame(f_value), aes(x=f_value)) +
  geom_histogram(color = 'black', fill = "red") +
  geom_rug()+ #Show the individual observations with black lines
  labs(title = "Frequency distribution of the f_value"
       ) +
  xlab("f_value") +
  ylab("Frequency ")
```

[ ]:
```
# Frequency distribution of the s_value
ggplot(data.frame(s_value), aes(x=s_value)) +
  geom_histogram(color = 'black', fill = "red") +
  geom_rug()+ #Show the individual observations with black lines
  labs(title = "Frequency distribution of the s_value"
       ) +
  xlab("s_value") +
  ylab("Frequency ")
```

[ ]:
```
# Create a data frame with the values of f_value and s_value and a column for␣
  ↪"group"
df <- data.frame(f_value, s_value, legend=rep(c("f_value","s_value"),␣
  ↪each=length(f_value)/2))

# Plot the scatter plot using and hiding legends
p <- ggplot(df, aes(x=f_value, y=s_value, color=legend)) +
  geom_point()+
  theme(legend.position="bottom")+ # show legend in bottom
  theme(legend.title = element_blank())+ # hide legend word
   #guides(color=FALSE)+ # Uncomment to hide legend
  ggtitle("Joint and Marginal Posterior Distribution")
#
# Show the plot
print(p)
```

[ ]:
```
# Create a data frame with the values of f_value and s_value and a column for␣
  ↪"group"
df <- data.frame(f_value, s_value, legend=rep(c("f_value","s_value"),␣
  ↪each=length(f_value)/2))

# Plot the scatter plot using and hiding legends
p <- ggplot(df, aes(x=f_value, y=s_value, color=legend)) +
  geom_point()+
  theme(legend.position="bottom")+
  theme(legend.title = element_blank())+
   #guides(color=FALSE)+ # Uncomment to hide legend
  ggtitle("Joint and Marginal Posterior Distribution")
#
# # Show the plot
# print(p)


# Marginal histograms by group
ggMarginal(p, type = "histogram",
          xparams = list(fill = 1),
           yparams = list(fill = 1))
```

[ ]:
```
```

[ ]:
```
```