**(7089CEM)**

**Introduction to Statistical Methods for Data Science**

MODULE LEADER: Dr. Fei He

Student Name: Siddhesh Ranjeet Desai

SID: 13008482

COURSEWORK

I can confirm that all work submitted is my own: Yes

Table of Contents

[Task 1: Preliminary Data Analysis 3](#_Toc128088147)

[Time series plots 3](#_Toc128088148)

[Distribution for each signal 7](#_Toc128088149)

[Correlation and scatter plots 10](#_Toc128088150)

[Task 2: Regression – modelling the relationship between audio and EEG signals 13](#_Toc128088151)

[Task 2.1: Estimate model parameter 13](#_Toc128088152)

[Task 2.2: Model residual (error) sum of squared errors (RSS) 16](#_Toc128088153)

[Task 2.3: Log-Likelihood function 19](#_Toc128088154)

[Task 2.4: Akaike information criterion (AIC) and Bayesian information criterion (BIC) 21](#_Toc128088155)

[Task 2.5: Distribution of model prediction errors (residuals) for each candidate model 23](#_Toc128088156)

[Task 2.6: Selecting the best regression model 27](#_Toc128088157)

[Task 2.7: Train and Test the selected best model 28](#_Toc128088158)

[Task 3: Approximate Bayesian Computation (ABC) 35](#_Toc128088159)

[Task 3.1 35](#_Toc128088160)

[Task 3.2 35](#_Toc128088161)

[Task 3.3 36](#_Toc128088162)

[Task 3.4 36](#_Toc128088163)

[Task 3.5 39](#_Toc128088164)

[References 40](#_Toc128088165)

[Appendix 42](#_Toc128088166)

All the tasks will be solved using the R programming language.

# Task 1: Preliminary Data Analysis

## Time series plots

##### Introduction

As a part of this task, we have to create time series plots of both EEG signals and audio signal. We will be making use of the ggplot() function of the R programming language which is part of the ggplot2 package to create time series plots. Since this package is not included by default we need to install it manually and load it into our project.



Code Block 1

Now that the package is imported we will be reading our data from the CSV files. The read.csv() function is used to read in CSV files, which are commonly used to store tabular data in a plain text format. The function creates a data frame from the CSV file, with the first row of the file usually assumed to be the header row.

A picture containing text

Description automatically generated

Code Block 2

ggplot() uses a data frame as input so we will be combining all the objects into a single data frame. We will be making use of the cbind() function to combine all the objects. The cbind() function is utilized to combine two data frames with equal number of rows. It can also add columns to vectors, matrices, or any data frame.



Code Block 3

Using the data frame we will now plot the Time Series graphs.

Graphical user interface, text

Description automatically generated

Code Block 4

The above code will create a line plot of the EEG signal ‘x1’ over time. The explanation of the code is as below

* **ggplot(data, aes(x = time, y = x1))** -> specifies the data to be plotted and the mapping of the data to aesthetics, in this case mapping the variable ‘time’ to the x-axis and ‘x1’ to the y-axis.
* **geom\_line(color = "blue", linewidth = 0.5)** -> adds the line to the plot, with the color parameter specifying the line ‘color’ and ‘linewidth’ parameter controlling the line thickness.
* The **‘ggtitle’** function sets the plot title, while **‘xlab’** and **‘ylab‘** set the x and y axis labels, respectively.
* **‘theme\_bw’** sets the plot theme to a black-and-white style, while the **‘theme’** function is used to customize various plot elements, including the **plot title, margins, and background color**.

Similarly, we will create a Time series plot of EEG signal ‘x2’ and sound signal ‘y’. The code for the same could be found in Appendix under the section “Time series plot of EEG signal 'x2'” and “Time series plot of sound signal 'y'” under Task 1.

##### Results

A picture containing chart

Description automatically generated

Time Series Plot 1: EEG signal 'x1'

A picture containing chart

Description automatically generated

Time Series Plot 2: EEG signal 'x2'

##### Chart, line chart Description automatically generated

Time Series Plot 3: Sound signal 'y'

##### Discussion

Let’s analyse and discuss the observation made from the plot using the below points.

* Trend – The overall trend seems to be stable for both the EEG signal ‘x1’ and ‘x2’ with few high points and few low points with time. The plot for sound signal ‘y’ seems to be fairly stable but has a considerable amount of deviations.
* Patterns – The EEG signal ‘x2’ can be seen spiking up after 25 seconds and falling drastically after 75 seconds. These points are the highest and the lowest point of the signal to be reached. There are frequent spike-ups and down in the sound signal ‘y’.

##### Conclusion

The plots of EEG signals ‘x1’ and ‘x2’ seem to be stable and continuous but the sound signal ‘y’ has a lot of fluctuations. This could be because of the additive noise in the signal.

## Distribution for each signal

##### Introduction

Distributions of each signal will be plotted using histograms. We will be again making use of the ggplot package to plot the distribution for both the EEG signals(x1,x2) and the sound signal(y).

Graphical user interface, text, application

Description automatically generated

Code Block 5

The above code will create a histogram of EEG signal ‘x1’. The explanation of the code is as below

* **ggplot(data, aes(x = x1))** -> The histogram is based on a data set called "data", with the x-axis mapped to a variable called "x1" using the aes() function.
* **geom\_histogram(binwidth = 0.2, fill = "black", color = "white")** -> Creates a histogram with customized properties like 'binwidth' which sets the width of the bins to 0.2, meaning that data values within a range of 0.2 will be grouped. The fill argument sets the color of the histogram bars to black, while the color argument sets the border color to white.

Similarly, we will create histograms of EEG signal ‘x2’ and sound signal ‘y’. The code for the same could be found in Appendix under the section “Histogram of EEG signal 'x2'” and “Histogram of sound signal 'y'” under Task 1.

##### Results

Chart, bar chart, histogram

Description automatically generated

Histogram 1: EEG signal 'x1'

Chart, bar chart, histogram

Description automatically generated

Histogram 2: EEG signal 'x2'

##### Chart, histogram Description automatically generated

Histogram 3: Sound signal 'y'

##### Discussion

* All three data values seem to show a good spread where the peak occurrences are at the centre of the graph.
* To create appropriate histograms the binwidth for EEG signals ‘x1’ and ‘x2’ was kept at 0.2. Whereas, for sound signal ‘y’, the binwidth is 2 since smaller segments are creating too many spikes.
* Neither of the graphs is either right-skewed or left-skewed.
* The no of outliers in all three cases is considerably low.
* Overall all three graphs seem to have a good fit distribution.

##### Conclusion

Overall, the graphs seem to have a good fit distribution, indicating that the data is well-distributed and there is no significant bias in the measurements.

## Correlation and scatter plots

##### Introduction

We will be computing the correlation as below



Code Block 6

Here,

* cor() -> It is a function used to calculate the correlation coefficient.
* round() -> This function is used to round the resulting decimal value to 2 decimal places.

Similarly, we can calculate the same for ‘x2’ to ‘y’. The code for the same could be found in Appendix under the section “Compute correlation of 'x2' to 'y'” under Task 1.

For scatter plots we will be again making use of the ggplot package.

Graphical user interface, text, application, chat or text message

Description automatically generated

Code Block 7

Above code will create a scatter plot between ‘x1’ and ‘y’. Here,

* **ggplot(data, aes(x = x1, y = y))** -> This function is used to initiate the plot, and the 'aes()' function is used to specify which variables to map to the x and y axis.
* **geom\_point()** -> This function is used to create the actual scatter plot by plotting each pair of corresponding values of the two variables as individual points.

Similarly, we can plot the same scatter plot for ‘x2’ and ‘y’. The code for the same could be found in Appendix under the section “Scatter plot between 'x2' brain signal and 'y' sound signal” under Task 1.

##### Results

Chart, line chart

Description automatically generated

Scatter Plot 1: 'x1' EGG signal and 'y' sound signal

Chart, scatter chart

Description automatically generated

Scatter Plot 2: 'x2' EGG signal and 'y' sound signal

##### Discussion

* EGG signal ‘x1’ and sound signal ‘y’ show a positive correlation with each other wherein we can see that as the value of ‘x1’ increases the value of sound signal ‘y’ also increases.
* EGG signal ‘x2’ and sound signal ‘y’ show a negative correlation with each other wherein the data points are scattered and do not show any correlation. This means as the value of ‘x2’ increases the sound signal ‘y’ tends to go down.

##### Conclusion

These observations imply that there is a relationship between the EGG signals and the sound signal and that this relationship can be both positive and negative depending on the signal being considered.

# Task 2: Regression – modelling the relationship between audio and EEG signals

## Task 2.1: Estimate model parameter

##### Introduction

In this task, we will be estimating model parameters for every candidate model given above using the Least Squares.

The first step to calculate this would be converting all the objects into a matrix. This could be achieved from the below code.

Text

Description automatically generated

Code Block 8

We will also be needing a matrix of one’s to compute θbias.



Code Block 9

The above code will create a matrix filled with ones, where the number of rows is equal to the number of rows in the matrix ‘X’ and the number of columns is 1.

Now that we have all the data let’s bind it together and create a design matrix for each candidate model. We will be making use of the cbind() function to concatenate all the columns.



Code Block 10

The above code will construct a linear regression model by combining the predictor variables in the matrix ‘X’ with additional polynomial features. Similarly, we can create a design matrix for all the other models specified. The code for the same could be found in Appendix under the section “Creating design matrix for each candidate model” under Task 2.1.

The next step would be to estimate the model parameters using least squares and pass all the data for training the model.



Code Block 11

The above code estimates the parameters of a linear regression model using least squares by fitting the model to the data using the design matrix model1.

Here,

* **t(model1)** -> takes the transpose of the matrix ‘model1’
* **%\*%** -> performs matrix multiplication
* **solve(t(model1) %\*% model1)** -> calculates the inverse of the square matrix obtained in the previous step

Similarly, we can estimate the model parameters for other models. The code for the same could be found in Appendix under the section “Estimate model parameters using least squares” under Task 2.1.

##### Results

All the values have been rounded off to 3 decimal values. Estimates of all the models are as below.

**Model 1**

|  |  |
| --- | --- |
| Parameters | y |
| x1^3 | 3.597 |
| x2^5 | -0.006 |
| Bias | -1.15 |

**Model 2**

|  |  |
| --- | --- |
| Parameters | y |
| x1^4 | 0.274 |
| x2^2 | 0.783 |
| Bias | -4.752 |

**Model 3**

|  |  |
| --- | --- |
| Parameters | y |
| x1^3 | 2.716 |
| x2 | -3.151 |
| x1 | 4.181 |
| Bias | -6.651 |

**Model 4**

|  |  |
| --- | --- |
| Parameters | y |
| x1 | 4.171 |
| x1^2 | 0.059 |
| x1^3 | 2.719 |
| x2^3 | -0.149 |
| Bias | -2.474 |

**Model 5**

|  |  |
| --- | --- |
| Parameters | y |
| x1^3 | 3.603 |
| x1^4 | -0.04 |
| x2 | -3.154 |
| Bias | -6.544 |

##### Discussion

All the model parameters were estimated using the least squares method. To print the output in a readable format I made use of row.names() and print() functions. Code for all the models could be found in the Appendix under the section “Print the estimated model parameters for each candidate model” in Task 2.1.

## Task 2.2: Model residual (error) sum of squared errors (RSS)

##### Introduction

As a part of this task, we will be calculating the model residual (error) sum of squared errors (RSS) for every candidate model using the model parameters calculated in the previous task.

To calculate the model residue we need to compute the values the model would predict after training it. As a first step, we will be creating a vector of predicted values.



Code Block 12

In the above code, all the predicted values will be stored in ‘y\_Hat1’. Similarly, we will be performing the same operation for all the other candidate models. Code for the same could be found in the Appendix under the section “Creating a vector of predicted values” in Task 2.2.

Once the predictions are done we will be computing the error or residual between the actual values and the predicted values.



Code Block 13

In the code above, the operation subtracts the predicted values of the response variable (y\_Hat1) from the actual values of the response variable (y) element-wise to obtain the residual values. This operation essentially measures the difference between the predicted and actual values of the response variable for each observation in the model.

The resulting ‘error1’ vector can be used to evaluate the performance of the regression model by calculating various measures of model fit or goodness-of-fit statistics, such as the sum of squared errors or the root mean squared error. The residuals can also be analyzed to identify patterns or trends in the model that may indicate problems with the assumptions of the regression analysis or opportunities for further model refinement.

Similarly, we will be calculating the error or residual for all the candidate models. Code for the same could be found in the Appendix under the section “Computing error or residual between the true response variable values (y) and the predicted values” in Task 2.2.

The final step is to calculate the residual sum of squares (RSS)



Code Block 14

In the above code, the operation calculates the sum of squared residuals (i.e., the sum of the squared differences between the predicted and actual values of the response variable) by first squaring each element in the error1 vector using the ^2 operator, and then adding up all the squared values using the sum() function.

Similarly, we will be calculating RSS values for all the candidate models. Code for the same could be found in the Appendix under the section “Calculating the residual sum of squares (RSS)” in Task 2.2.

##### Results

|  |  |
| --- | --- |
| RSS | Value |
| Model 1 | 30743.059 |
| Model 2 | 438230.382 |
| Model 3 | 1525.623 |
| Model 4 | 7949.508 |
| Model 5 | 17138.144 |

##### Discussion

When evaluating the performance of a statistical model there are various factors that are needed to be considered before making a decision. One such factor is the Residual Sum of Squares (RSS). Two main key components to check when evaluating using RSS are the no of variables of a model and which model has the lowest RSS value.

We evaluated 5 models as part of this task and the observations are as below

* Model 1 had 2 variables and has the second-largest RSS score.
* Model 2 also had 2 variables but has the highest RSS score of all the models.
* Model 3 had 3 variables and had the least RSS score.
* Model 4 had 4 variables and had the second least RSS score.
* Model 5 had 3 variables and had the third-largest RSS score.

##### Conclusion

Based on the information provided, it appears that Model 3, which had the least RSS score, was the best model of the five evaluated. This model had three variables, indicating that it may strike a good balance between complexity and fit. However, it's important to note that RSS alone does not provide a complete picture of model performance, and other metrics and factors should also be considered when selecting the best model. Additionally, further analysis may be needed to understand the relationships between variables and the specific context of the problem being studied.

## Task 2.3: Log-Likelihood function

##### Introduction

To calculate log-likelihood we have to calculate sample variance. We can calculate the sample variance as below



Code Block 15

Here,

* **sigma\_1** -> name of the variable that will store the standard deviation or scale parameter of the regression model.
* **rss1** -> residual sum of squares of the regression model, which is calculated in the previous step.
* **length(data$x1)** -> number of observations or data points in the regression model for the predictor variable x1.

Similarly, we can calculate the sample variance of other candidate models. The code for the same could be found in Appendix under the section “Calculating the sample variance of x1” of Task 2.3.

Now that we have our sample variance we can calculate the log-likelihood value as below



Code Block 16

Here,

* **logLik1** -> stores the log-likelihood value of the regression model.
* **length(data$x1)** -> number of observations or data points in the regression model for the predictor variable x1.
* **pi** -> constant in R that represents the mathematical value of pi.
* **sigma\_1** -> standard deviation or scale parameter of the regression model, which is calculated in the previous step.
* **rss1** -> residual sum of squares of the regression model, which is also calculated in a previous step.
* **log()** is a function in R that calculates the natural logarithm of a value.

Similarly, we can calculate the log-likelihood of other candidate models. The code for the same could be found in Appendix under the section “Calculating log-likelihood” of Task 2.3.

##### Results

|  |  |  |
| --- | --- | --- |
| Models | Sample Variance (sigma) | Log-likelihood |
| Model 1 | 12.815 | -6465.687 |
| Model 2 | 182.672 | -9654.184 |
| Model 3 | 0.636 | -2861.774 |
| Model 4 | 3.314 | -4842.622 |
| Model 5 | 7.144 | -5764.458 |

##### Discussion

When evaluating a model using the log-likelihood method it is important to check which value is the highest. It is noteworthy that including more predictor variables in a model typically results in an increase in the log-likelihood value, even if the extra predictor variables lack statistical significance.

Based on the calculation the observation was,

* Model 3 had the highest value.
* Although Models 4 and 5 had more predictive variables still their log-likelihood values are considerably low.
* Models 1 and 2 also have low scores.

##### Conclusion

According to log-likelihood values, Model 3 seems to fit perfectly.

## Task 2.4: Akaike information criterion (AIC) and Bayesian information criterion (BIC)

##### Introduction

Akaike information criterion (AIC) can be computed as below



Code Block 17

Here,

* **logLik1** -> represents the maximum log-likelihood of the model.
* **length(thetaHat1)** -> gives the number of estimated parameters in the model.

Similarly, we can calculate the Akaike information criterion (AIC) of other candidate models. The code for the same could be found in Appendix under the section “Akaike information criterion (AIC)” of Task 2.4.

Bayesian information criterion (BIC) can be computed as below



Code Block 18

Here,

* **length(data$x1)** -> number of observations in the input variable.
* **logLik1** -> represents the maximum log-likelihood of the model.

Similarly, we can calculate the Bayesian information criterion (BIC) of other candidate models. The code for the same could be found in Appendix under the section “Bayesian information criterion (BIC)” of Task 2.4.

##### Results

|  |  |  |
| --- | --- | --- |
| Models | Akaike information criterion (AIC) | Bayesian information criterion (BIC) |
| Model 1 | 12937.374 | 12954.724 |
| Model 2 | 19314.368 | 19331.717 |
| Model 3 | 5731.548 | 5754.68 |
| Model 4 | 9695.244 | 9724.161 |
| Model 5 | 11536.916 | 11560.049 |

##### Discussion

Both the Akaike information criterion (AIC) and Bayesian information criterion (BIC) prove to be efficient methods in determining if the model is a good fit or not. For a model to be a good fit both the AIC and the BIC values need to be on the lower side.

After computing both the values for each of our candidate models we can observe that,

* Models 1, 2 and 3 have significantly bigger values for both operations.
* Models 3 and 4 have lower numbers but Model 3 has the lowest.

##### Conclusion

Based on the AIC and the BIC values Model 3 seem to be a better-fitting algorithm.

## Task 2.5: Distribution of model prediction errors (residuals) for each candidate model

##### Introduction

As a part of this task, we will be plotting the error distribution and would be evaluating if those distributions are close to Normal/Gaussian.   
  
We will be plotting residuals using QQ-plot as below

Text

Description automatically generated

Code Block 19

Here,

* **qqnorm(error1, main = "Q-Q Plot for Model 1")** -> creates the Q-Q plot by plotting the ordered values of the residuals error1 against the theoretical quantiles of a standard normal distribution.
* **qqline(error1)** -> adds a reference line to the plot that indicates the expected values of the residuals if they were normally distributed.

Similarly, we will generate plots for other candidate models. Code for the same could be found in Appendix under the section “Task 2.5: Distribution of model prediction errors (residuals) for each candidate model” in Task 2.5.

Results  
  
Chart, line chart

Description automatically generated

Q-Q Plot 1: Model 1

Chart, line chart

Description automatically generated

Q-Q Plot 2: Model 2

##### Chart, line chart Description automatically generated

Q-Q Plot 3: Model 3

Chart, line chart

Description automatically generated

Q-Q Plot 4: Model 4

Chart, line chart

Description automatically generated

Q-Q Plot 5: Model 5

##### Discussion

An ideal model would be the one whose error points pass through the reference line. From the Q-Q plots which we plotted we can observe that

* Model 1 plot follows the reference line to a significant amount but there are certain deviations and are at extreme points as well.
* Model 2 plot doesn’t follow the reference line and hence is not distributed properly.
* Model 3 plot follows the reference line and all the point fall on the reference line. There is a single point which is out of scope.
* Model 4 plot hardly follows the reference line.
* Model 5 plot goes zigzag and some points fall on the reference line.

##### Conclusion

Based on the Q-Q plots we can conclude that Model 3 is fitting properly over the reference line.

## Task 2.6: Selecting the best regression model

##### Introduction

As part of this task, we need to select the ‘best’ regression model considering all the aspects like AIC, BIC and distribution of model residuals which we calculated as previous tasks for all the five candidate models.

##### Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Models | Number of parameters | RSS | Sample Variance (sigma) | Log-likelihood | Akaike information criterion (AIC) | Bayesian information criterion (BIC) |
| Model 1 | 2 | 30743.059 | 12.815 | -6465.687 | 12937.374 | 12954.724 |
| Model 2 | 2 | 438230.382 | 182.672 | -9654.184 | 19314.368 | 19331.717 |
| Model 3 | 3 | 1525.623 | 0.636 | -2861.774 | 5731.548 | 5754.68 |
| Model 4 | 4 | 7949.508 | 3.314 | -4842.622 | 9695.244 | 9724.161 |
| Model 5 | 3 | 17138.144 | 7.144 | -5764.458 | 11536.916 | 11560.049 |

##### Discussion

As seen in the previous tasks Model 3 checkmarks in all the aspects to be a best fitting regression model of all the candidate models.

* It has 3 parameters which are not the maximum no of parameters as no of parameters tends to increase the complexity in predicting.
* It has the lowest RSS value.
* Has the smallest Sample Variance.
* Log-likelihood value is also the lowest.
* AIC and BIC values too are the smallest compared to other models.
* Was the best fitting with reference to error plot distribution.

##### Conclusion

Model 3 is the best regression model of five of the candidate models provided.

## Task 2.7: Train and Test the selected best model

##### Introduction

From the previous tasks that we performed, we were successfully able to evaluate that Model 3 was the best-fitting model of all the 5 candidate models. Now, as part of this task, we will be performing the below tasks on the best model which we have chosen.

* Estimating model parameters using the training dataset
* Computing the model’s output/prediction on the testing data
* Computing the 95% (model prediction) confidence intervals and plotting them (with error bars) together with the model prediction, as well as the testing data samples.

Let us start with some prerequisites before performing the above operations.

We will be again reading the data from CSV files. But this time we will be only using the

EEG signal and the sound signal file. Also, we will create a data frame by binding both values and a vector of integers that represents the row indices of the dataframe we created by binding the values.

Graphical user interface, text

Description automatically generated

Code Block 20

The next step is to split the dataframe into two parts for training and testing the model. The spit has to be 70-30 where 70% of the data is used to train the model and the remaining 30% is used to test the model predictions. To do this we will first split the indices into 70-30 using sample() and setdiff(). Once we have the indices we will be using them to create our training and test datasets.

Graphical user interface, text

Description automatically generated

Code Block 21

Here,

* **sample()** -> is used to randomly select 70% of the indices.
* **floor(0.7 \* length(row\_length))** -> argument specifies the number of indices to select, which is 70% of the total number of rows, rounded down to the nearest integer.
* **setdiff()** -> is used to find the set difference between the **row\_length** vector and the **training\_rows** vector. This means that it will select all the indices in **row\_length** that are not in **training\_rows**, which will give us the remaining 30% of row indices for the test set.

Now that we have two separate datasets for training and testing we will further split them into their object form and also create ones matrix for both training and testing.

Graphical user interface, text

Description automatically generated

Code Block 22

Task 2.7.1: Estimating model parameters using the training dataset

In this task, we will be training and testing our best-fitting model using the training and test datasets and predicting the values.

Graphical user interface, text, application, email

Description automatically generated

Code Block 23

Task 2.7.2: Computing the model’s output/prediction on the testing data

We will be now computing the model’s prediction of the testing data. To calculate that we will first compute the y\_hat value i.e. the predicted values, then we will be comparing the actual and predicted values to find the error/residual and using that we will get the sum of squared errors(SSE) values.

Graphical user interface, text, application

Description automatically generated

Code Block 24

Task 2.7.3: Computing the 95% (model prediction) confidence intervals and plotting them (with error bars) together with the model prediction, as well as the testing data samples

We will be calculating a 95% confidence interval as part of this task.

Graphical user interface, text, application, email

Description automatically generated

Code Block 25

Here,

* **model\_data = matrix(train\_model[i, ], 1, 4)** -> The code is extracting a single row from the training dataset train\_model and stores it in a new matrix called model\_data.
* **zero\_hat[i,1] = model\_data %\*% ctheta %\*% t(model\_data)** -> The code is using the model\_data matrix and the inverse of the cross-product (ctheta) to compute the predicted variance for the i-th row of the training dataset. The resulting value is stored in the corresponding row of the zero\_hat matrix.
* **CI = 2 \* sqrt(zero\_hat)** -> The code is computing the confidence interval by multiplying the square root of the predicted variance by 2 and storing it in the variable CI.
* **upper = y\_hat + CI and lower = y\_hat - CI** -> The code is calculating the upper and lower limits of the confidence interval by adding and subtracting the confidence interval from the predicted values (stored in y\_hat).

Now let’s plot model predictions with error bars and also show the upper and lower limit as calculated. We will be using ggplot to plot the graph and the code would be as below

Graphical user interface, text, application

Description automatically generated

Code Block 26

##### Results

Task 2.7.1: Estimating model parameters using the training dataset

|  |  |  |
| --- | --- | --- |
| Model Parameters | train\_thetaHat | test\_thetaHat |
| x1^3 | 2.715525 | 2.715069 |
| x2 | -3.159385 | -3.132432 |
| x1 | 4.182633 | 4.178275 |
| Bias | -6.648339 | -6.658613 |

Task 2.7.2: Computing the model’s output/prediction on the testing data

"**Sum of Squared Errors (SSE):** 470.328"

Task 2.7.3: Computing the 95% (model prediction) confidence intervals and plotting them (with error bars) together with the model prediction, as well as the testing data samples

Chart, box and whisker chart

Description automatically generated

Figure 1: 95% Confidence Intervals with error bars

##### Discussion

* The data was split into 70-30 to form training and testing dataframes.
* The model was trained using the training set.
* The actual and predicted values were almost the same with minor deviations for each of the parameters.
* The 95% confidence interval graph showed the predicted values and their upper and lower limit.

##### Conclusion

The model that has been trained using the training dataset has been able to appropriately predict the values as expected, demonstrating its effectiveness and efficiency in achieving the intended purpose.

# Task 3: Approximate Bayesian Computation (ABC)

##### Introduction

## Task 3.1

We need to compute 2 parameter posterior distributions with the largest absolute values from the least absolute values which we calculated in Task 2.1 for Model 3. All the other parameters will be fixed as constants.

A screenshot of a computer

Description automatically generated with medium confidence

Code Block 27

## Task 3.2

By using a uniform distribution as prior around the estimated parameter values for theta3 and theta\_bias we need to determine the range of the prior distribution.

Text

Description automatically generated

Code Block 28

## Task 3.3

We need to draw samples from the above uniform prior and perform rejection ABC using theta3 and theta\_bias.

Graphical user interface, text, application, email

Description automatically generated

Code Block 29

## Task 3.4

We will be plotting the joint and marginal posterior distribution for theta3 and theta\_bias.

Bayesian statistics employs two probability distributions, namely joint and marginal posterior distributions, to represent the uncertainty of model parameters following the inclusion of prior knowledge and observed data.

Text

Description automatically generated

Code Block 30

##### Results

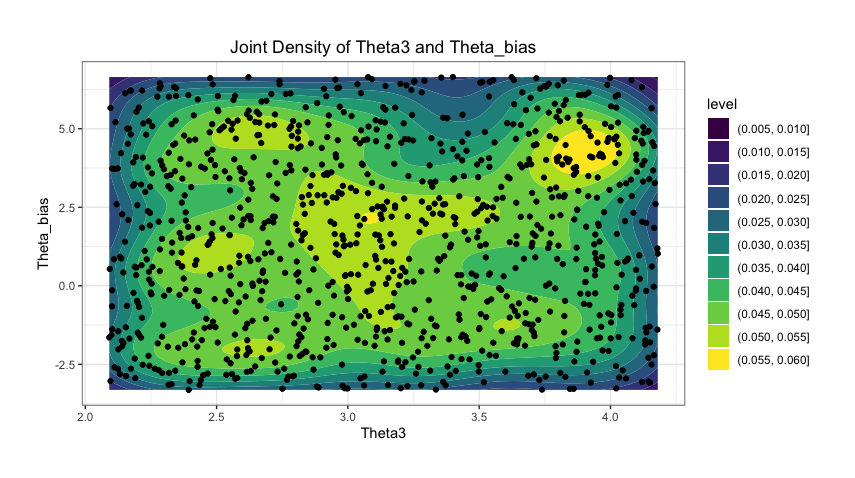


Figure 2: Joint and marginal posterior distribution of theta3 and theta\_bias

Chart, bar chart, histogram

Description automatically generated

Histogram 4: Marginal distribution of theta3

Chart, bar chart, histogram

Description automatically generated

Histogram 5: Marginal distribution of theta\_bias

## Task 3.5

##### Discussion

* Using the ‘rejection ABC’ method we were able to compute the posterior distribution of the two largest value parameters of model 3.
* The joint density plot of theta3 and theta\_bias shows around 2000 values predicted by the model.
* The marginal distribution of theta3 and theta\_bias shows uniform distribution of data across the range.
* Neither of the data is skewed at any of the ends.
* There seem to be a few outliers.

##### Conclusion

All the tasks specified were completed by implementing Approximate Bayesian Computation (ABC) on Model 3.

# References

1. Patil, P. (2022) *What is exploratory data analysis?*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15 (Accessed: February 14, 2023).
2. *Function reference* (no date) *• ggplot2*. Available at: https://ggplot2.tidyverse.org/reference/ (Accessed: February 17, 2023).
3. What is cbind in R? (no date) ProjectPro. Available at: https://www.projectpro.io/recipes/what-is-cbind-r# (Accessed: February 20, 2023).
4. Contributor, T.T. (2018) *What is time series chart?: Definition from TechTarget*, *WhatIs.com*. TechTarget. Available at: https://www.techtarget.com/whatis/definition/time-series-chart# (Accessed: February 20, 2023).
5. R - histograms (no date) Tutorials Point. Available at: https://www.tutorialspoint.com/r/r\_histograms.htm (Accessed: February 20, 2023).
6. Interpret the key results for histogram (no date) Minitab. Available at: https://support.minitab.com/en-us/minitab/20/help-and-how-to/graphs/histogram/interpret-the-results/key-results/ (Accessed: February 21, 2023).
7. *Correlation* (no date) *JMP*. Available at: https://www.jmp.com/en\_gb/statistics-knowledge-portal/what-is-correlation.html (Accessed: February 21, 2023).
8. Kenton, W. (2023) Least squares method: What it means, how to use it, with examples, Investopedia. Investopedia. Available at: https://www.investopedia.com/terms/l/least-squares-method.asp (Accessed: February 21, 2023).
9. Barone, A. (2022) Residual sum of squares (RSS): What it is, how to calculate it, Investopedia. Investopedia. Available at: https://www.investopedia.com/terms/r/residual-sum-of-squares.asp# (Accessed: February 22, 2023).
10. Zach (2021) *How to interpret log-likelihood values (with examples)*, *Statology*. Available at: https://www.statology.org/interpret-log-likelihood/ (Accessed: February 22, 2023).
11. Bevans, R. (2022) *Akaike information criterion: When & how to use it (example)*, *Scribbr*. Available at: https://www.scribbr.com/statistics/akaike-information-criterion/ (Accessed: February 22, 2023).
12. Datalab, A. (2019) *What is Bayesian Information Criterion (BIC)?*, *Medium*. Medium. Available at: https://medium.com/@analyttica/what-is-bayesian-information-criterion-bic-b3396a894be6 (Accessed: February 22, 2023).
13. Chen, J. (2023) Normal distribution: What it is, properties, uses, and Formula, Investopedia. Investopedia. Available at: https://www.investopedia.com/terms/n/normaldistribution.asp (Accessed: February 22, 2023).
14. Hartig, F. *et al.* (2014) *Explaining the ABC-rejection algorithm in R*, *theoretical ecology*. Available at: https://theoreticalecology.wordpress.com/2014/06/02/explaining-the-abc-rejection-algorithm-in-r/ (Accessed: February 23, 2023).

# Appendix

#### R Script to perform all the above tasks.

########################################################

# Importing all the data files #

########################################################

# This is used to load the ggplot2 package into the current working directory

library(ggplot2)

# Loading data from respective CSV files and storing into data frame objects

X = read.csv("X.csv")

y = read.csv("y.csv")

time = read.csv("time.csv")

# Combining all the objects into a data frame

data = cbind(X, y, time)

# Verifying if the data frame is created appropriately

head(data)

########################################################

# Task 1: Preliminary data analysis #

########################################################

## Time series plots (of audio and EEG signals)

# Time series plot of EEG signal 'x1'

ggplot(data, aes(x = time, y = x1)) +

geom\_line(color = "blue", linewidth = 0.5) +

ggtitle("Time series plot of EEG signal 'x1'") +

xlab("Time(s)") +

ylab("'x1' Signal(Hz)") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

# Time series plot of EEG signal 'x2'

ggplot(data, aes(x = time, y = x2)) +

geom\_line(color = "blue", linewidth = 0.5) +

ggtitle("Time series plot of EEG signal 'x2'") +

xlab("Time(s)") +

ylab("'x2' Signal(Hz)") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

# Time series plot of sound signal 'y'

ggplot(data, aes(x = time, y = y)) +

geom\_line(color = "black", linewidth = 0.5) +

ggtitle("Time series plot of sound signal 'y'") +

xlab("Time(s)") +

ylab("Sound signal 'y'(Hz)") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

## Distribution for each signal

# Histogram of EEG signal 'x1'

ggplot(data, aes(x = x1)) +

geom\_histogram(binwidth = 0.2, fill = "black", color = "white") +

xlab("EEG signal 'x1'") +

ylab("Frequency") +

ggtitle("Histogram of EEG signal 'x1'") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

# Histogram of EEG signal 'x2'

ggplot(data, aes(x = x2)) +

geom\_histogram(binwidth = 0.2, fill = "black", color = "white") +

xlab("EEG signal 'x2'") +

ylab("Frequency") +

ggtitle("Histogram of EEG signal 'x2'") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

# Histogram of sound signal 'y'

ggplot(data, aes(x = y)) +

geom\_histogram(binwidth = 2, fill = "black", color = "white") +

xlab("Sound signal 'y'") +

ylab("Frequency") +

ggtitle("Histogram of sound signal 'y'") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

## Correlation and scatter plots (between the audio and brain signals)

# Compute correlation of 'x1' to 'y'

cor\_val1 = round(cor(data$x1, data$y), 2)

# Scatter plot between 'x1' brain signal and 'y' sound signal

ggplot(data, aes(x = x1, y = y)) +

geom\_point() +

xlab("'x1' Signal(Hz)") +

ylab("Sound signal 'y'(Hz)") +

ggtitle("Scatter plot between 'x1' brain signal and 'y' sound signal") +

annotate("text", x = max(data$x1), y = max(data$y), label = paste0("correlation = ", cor\_val1), hjust = 3.5, vjust = 1) +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

# Compute correlation of 'x2' to 'y'

cor\_val2 = round(cor(data$x2, data$y), 2)

# Scatter plot between 'x2' brain signal and 'y' sound signal

ggplot(data, aes(x = x2, y = y)) +

geom\_point() +

xlab("'x2' Signal(Hz)") +

ylab("Sound signal 'y'(Hz)") +

ggtitle("Scatter plot between 'x2' brain signal and 'y' sound signal") +

annotate("text", x = max(data$x2), y = max(data$y), label = paste0("correlation = ", cor\_val2), hjust = 1, vjust = 1) +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

#######################################################################

# Task 2: Regression - modelling the relationship between audio and EEG signals

#######################################################################

# Converting objects into a matrix

X = data.matrix(X)

y = data.matrix(y)

time = data.matrix(time)

# Creating ones matrix

ones = matrix(1 , length(X[,1]),1)

## Task 2.1: Estimating model parameters 𝜽={𝜃1,𝜃2,⋯,𝜃𝑏𝑖𝑎𝑠}𝑇 for every candidate model using Least Squares (𝜽̂= (𝐗𝑇𝐗)−1𝐗𝑇𝐲)

# Creating a design matrix for each candidate model

model1 = cbind(X[,1]^3, X[,2]^5, ones)

model2 = cbind(X[,1]^4, X[,2]^2, ones)

model3 = cbind(X[,1]^3, X[,2], X[,1], ones)

model4 = cbind(X[,1], X[,1]^2, X[,1]^3, X[,2]^3, ones)

model5 = cbind(X[,1]^3, X[,1]^4, X[,2], ones)

# Estimate model parameters using least squares

thetaHat1 = solve(t(model1) %\*% model1) %\*% t(model1) %\*% y

thetaHat2 = solve(t(model2) %\*% model2) %\*% t(model2) %\*% y

thetaHat3 = solve(t(model3) %\*% model3) %\*% t(model3) %\*% y

thetaHat4 = solve(t(model4) %\*% model4) %\*% t(model4) %\*% y

thetaHat5 = solve(t(model5) %\*% model5) %\*% t(model5) %\*% y

# Print the estimated model parameters for each candidate model

# Model 1

row\_names = c("x1^3","x2^5","Bias")

row.names(thetaHat1) = row\_names

thetaHat1 = round(thetaHat1, 3)

print(thetaHat1)

# Model 2

row\_names = c("x1^4","x2^2","Bias")

row.names(thetaHat2) = row\_names

thetaHat2 = round(thetaHat2, 3)

print(thetaHat2)

# Model 3

row\_names = c("x1^3","x2","x1","Bias")

row.names(thetaHat3) = row\_names

thetaHat3 = round(thetaHat3, 3)

print(thetaHat3)

# Model 4

row\_names = c("x1","x1^2","x1^3","x2^3","Bias")

row.names(thetaHat4) = row\_names

thetaHat4 = round(thetaHat4, 3)

print(thetaHat4)

# Model 5

row\_names = c("x1^3","x1^4","x2","Bias")

row.names(thetaHat5) = row\_names

thetaHat5 = round(thetaHat5, 3)

print(thetaHat5)

## Task 2.2: Computing the model residual (error) sum of squared errors (RSS)

# Creating a vector of predicted values

y\_Hat1 = model1 %\*% thetaHat1

y\_Hat2 = model2 %\*% thetaHat2

y\_Hat3 = model3 %\*% thetaHat3

y\_Hat4 = model4 %\*% thetaHat4

y\_Hat5 = model5 %\*% thetaHat5

# Computing error or residual between the true response variable values (y) and the predicted values

error1 = y - y\_Hat1

error2 = y - y\_Hat2

error3 = y - y\_Hat3

error4 = y - y\_Hat4

error5 = y - y\_Hat5

# Calculating the residual sum of squares (RSS)

rss1 = sum(error1^2)

rss2 = sum(error2^2)

rss3 = sum(error3^2)

rss4 = sum(error4^2)

rss5 = sum(error5^2)

# Printing residual sum of squares (RSS)

print(paste("rss1:", toString(round(rss1, 3))))

print(paste("rss2:", toString(round(rss2, 3))))

print(paste("rss3:", toString(round(rss3, 3))))

print(paste("rss4:", toString(round(rss4, 3))))

print(paste("rss5:", toString(round(rss5, 3))))

## Task 2.3: Computing the log-likelihood function for every candidate model

# Calculating the sample variance of x1

sigma\_1 = rss1/(length(data$x1)-1)

sigma\_2 = rss2/(length(data$x1)-1)

sigma\_3 = rss3/(length(data$x1)-1)

sigma\_4 = rss4/(length(data$x1)-1)

sigma\_5 = rss5/(length(data$x1)-1)

# Calculating log-likelihood

logLik1 = -(length(data$x1)/2) \* log(2 \* pi) - (length(data$x1)/2) \* log(sigma\_1) - (1/(2 \* sigma\_1)) \* rss1

logLik2 = -(length(data$x1)/2) \* log(2 \* pi) - (length(data$x1)/2) \* log(sigma\_2) - (1/(2 \* sigma\_2)) \* rss2

logLik3 = -(length(data$x1)/2) \* log(2 \* pi) - (length(data$x1)/2) \* log(sigma\_3) - (1/(2 \* sigma\_3)) \* rss3

logLik4 = -(length(data$x1)/2) \* log(2 \* pi) - (length(data$x1)/2) \* log(sigma\_4) - (1/(2 \* sigma\_4)) \* rss4

logLik5 = -(length(data$x1)/2) \* log(2 \* pi) - (length(data$x1)/2) \* log(sigma\_5) - (1/(2 \* sigma\_5)) \* rss5

# Printing sample variance and log-likelihood

print(paste("sigma\_1:", toString(round(sigma\_1, 3)), ", logLik1:", toString(round(logLik1, 3)), sep = " "))

print(paste("sigma\_2:", toString(round(sigma\_2, 3)), ", logLik2:", toString(round(logLik2, 3)), sep = " "))

print(paste("sigma\_3:", toString(round(sigma\_3, 3)), ", logLik3:", toString(round(logLik3, 3)), sep = " "))

print(paste("sigma\_4:", toString(round(sigma\_4, 3)), ", logLik4:", toString(round(logLik4, 3)), sep = " "))

print(paste("sigma\_5:", toString(round(sigma\_5, 3)), ", logLik5:", toString(round(logLik5, 3)), sep = " "))

## Task 2.4: Computing the Akaike information criterion (AIC) and Bayesian information criterion (BIC) for every candidate model

# Akaike information criterion (AIC)

aic1 = 2 \* length(thetaHat1) - 2 \* logLik1

aic2 = 2 \* length(thetaHat2) - 2 \* logLik2

aic3 = 2 \* length(thetaHat3) - 2 \* logLik3

aic4 = 2 \* length(thetaHat4) - 2 \* logLik4

aic5 = 2 \* length(thetaHat5) - 2 \* logLik5

# Bayesian information criterion (BIC)

bic1 = 3 \* log(length(data$x1)) - 2 \* logLik1

bic2 = 3 \* log(length(data$x1)) - 2 \* logLik2

bic3 = 4 \* log(length(data$x1)) - 2 \* logLik3

bic4 = 5 \* log(length(data$x1)) - 2 \* logLik4

bic5 = 4 \* log(length(data$x1)) - 2 \* logLik5

# Printing Akaike information criterion (AIC) and Bayesian information criterion (BIC) values

print(paste("AIC\_1:", toString(round(aic1, 3)), ", BIC\_1:", toString(round(bic1, 3)), sep = " "))

print(paste("AIC\_2:", toString(round(aic2, 3)), ", BIC\_2:", toString(round(bic2, 3)), sep = " "))

print(paste("AIC\_3:", toString(round(aic3, 3)), ", BIC\_3:", toString(round(bic3, 3)), sep = " "))

print(paste("AIC\_4:", toString(round(aic4, 3)), ", BIC\_4:", toString(round(bic4, 3)), sep = " "))

print(paste("AIC\_5:", toString(round(aic5, 3)), ", BIC\_5:", toString(round(bic5, 3)), sep = " "))

## Task 2.5: Distribution of model prediction errors (residuals) for each candidate model

# Generate QQ plot with error1 data for Model 1

qqnorm(error1, main = "Q-Q Plot for Model 1")

# Add a horizontal line indicating the expected values

qqline(error1)

# Generate QQ plot with error2 data for Model 2

qqnorm(error2, main = "Q-Q Plot for Model 2")

# Add a horizontal line indicating the expected values

qqline(error2)

# Generate QQ plot with error3 data for Model 3

qqnorm(error3, main = "Q-Q Plot for Model 3")

# Add a horizontal line indicating the expected values

qqline(error3)

# Generate QQ plot with error4 data for Model 4

qqnorm(error4, main = "Q-Q Plot for Model 4")

# Add a horizontal line indicating the expected values

qqline(error4)

# Generate QQ plot with error5 data for Model 5

qqnorm(error5, main = "Q-Q Plot for Model 5")

# Add a horizontal line indicating the expected values

qqline(error5)

## Task 2.7

# Reading the data files

X = read.csv("X.csv")

y = read.csv("y.csv")

# Creating a data frame by binding both the datasets

dataframe = cbind(X,y)

# Create a vector of indices corresponding to each observation in your dataset

row\_length = 1:nrow(dataframe)

# Use the sample function to randomly select 70% of the indices for the training set

training\_rows = sample(row\_length, floor(0.7 \* length(row\_length)))

# Use the setdiff function to select the remaining 30% of indices for the test set

test\_rows = setdiff(row\_length, training\_rows)

# Use the training\_indices and test\_indices vectors to create your training and test datasets

training\_set = dataframe[training\_rows, ]

test\_set = dataframe[test\_rows, ]

# Printing the dimensions of training\_set and test\_set

dim(training\_set)

dim(test\_set)

# Splitting the data into training data set

x1\_train = training\_set$x1

x2\_train = training\_set$x2

y\_train = training\_set$y

# Splitting the data into testing data set

x1\_test = test\_set$x1

x2\_test = test\_set$x2

y\_test = test\_set$y

# Creating ones matrix for training and testing data set

ones\_train = matrix(1 , length(training\_rows),1)

ones\_test = matrix(1 , length(test\_rows),1)

## Task 2.7.1: Estimating model parameters using the training dataset

# Train the model

train\_model = cbind(x1\_train^3, x2\_train, x1\_train, ones\_train)

train\_thetaHat = solve(t(train\_model) %\*% train\_model) %\*% t(train\_model) %\*% y\_train

row\_names = c("x1^3","x2","x1","Bias")

row.names(train\_thetaHat) = row\_names

train\_thetaHat

# Testing and predicting the values

test\_model = cbind(x1\_test^3, x2\_test, x1\_test, ones\_test)

test\_thetaHat = solve(t(test\_model) %\*% test\_model) %\*% t(test\_model) %\*% y\_test

row\_names = c("x1^3","x2","x1","Bias")

row.names(test\_thetaHat) = row\_names

test\_thetaHat

## Task 2.7.2: Computing the model‚Äôs output/prediction on the testing data

# Calculating the predicted values

y\_hat = test\_model %\*% train\_thetaHat

# Calculating the difference between the actual values and the predicted values

error = y\_test - y\_hat

# Calculating the sum of squared errors (SSE) between the predicted values and the actual values of the test dataset

sse = norm(error , type = "2")^2

# Printing the value of sum of squared errors (SSE)

print(paste("Sum of Squared Errors (SSE):", toString(round(sse, 3))))

## Task 2.7.3: Computing the 95% (model prediction) confidence intervals and plotting them (with error bars) together with the model prediction, as well as the testing data samples

# Storing the number of rows of y\_hat

n = nrow(y\_hat)

# Computing sample variance

variance = sse/(n-1)

# Computing the inverse of the cross-product

ctheta = (solve(t(test\_model) %\*% test\_model))

# Creating a zero matrix

zero\_hat = matrix(0, n , 1)

# Computing predicted variance

for (i in 1:n) {

model\_data = matrix(train\_model[i, ], 1, 4)

zero\_hat[i,1] = model\_data %\*% ctheta %\*% t(model\_data)

}

# Calculating the Confidence Interval

CI = 2 \* sqrt(zero\_hat)

print(paste("Confidence Interval:", toString(round(CI, 3))))

# Calculating the upper and lower limits of the Confidence Interval

upper = y\_hat + CI

lower = y\_hat - CI

# Creating data frame combining all the values

df = data.frame(x = 1:length(y\_test), y\_test = y\_test, y\_hat = y\_hat,

lower\_bound = lower, upper\_bound = upper)

# Plotting model prediction with error bars

ggplot(df, aes(x = x)) +

geom\_line(aes(y = y\_test)) +

geom\_point(aes(y = y\_hat), color = "blue") +

geom\_segment(aes(x = x, y = lower, xend = x, yend = upper),

size = 10) +

ylim(range(c(y\_test, y\_hat, upper, lower))) +

ggtitle("95% (model prediction) Confidence Intervals with error bars") +

xlab("Time(s)") +

ylab("y prediction") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

###################################################################

# Task 3: Approximate Bayesian Computation (ABC) #

###################################################################

## Task 3.1

# Combining X and y objects into a data frame

data = cbind(X,y)

# Extracting values from thetaHat3

theta2 = thetaHat3[2,]

theta3 = thetaHat3[3,]

theta\_bias = thetaHat3[4,]

# Task 3.2

# Creating random numbers from a uniform distribution

theta3\_prior = runif(1000, min = 0.5 \* theta3, max = 1 \* theta3)

theta\_bias\_prior = runif(1000, min = -0.5 \* abs(theta\_bias), max = 1 \* abs(theta\_bias))

# Creating empty vectors

accepted\_samples\_3 = c()

accepted\_samples\_bias = c()

# Task 3.3

# Compute predicted values and error

for (i in 1:length(theta3\_prior)) {

y\_pred = theta3\_prior[i] \* data$x1 ^ 3 + theta2 \* data$x2 + theta3\_prior[i] \* data$x1 + theta\_bias\_prior[i]

pred\_error = data$y - y\_pred

rss = sum(pred\_error ^ 2)

if (rss < 300000) {

accepted\_samples\_3 = c(accepted\_samples\_3, theta3\_prior[i])

accepted\_samples\_bias = c(accepted\_samples\_bias, theta\_bias\_prior[i])

}

}

# Defining a function to removes missing values (NA's)

remove\_NA = function(x) {

x[!is.na(x)]

}

# Remove missing values from accepted samples vectors

accepted\_samples\_3 = remove\_NA(accepted\_samples\_3)

accepted\_samples\_bias = remove\_NA(accepted\_samples\_bias)

# Task 3.4

# Plotting Joint Density of Theta3 and Theta\_bias

ggplot(data = data.frame(theta3\_prior, theta\_bias\_prior)) +

geom\_density\_2d\_filled(mapping = aes(x = theta3\_prior, y = theta\_bias\_prior)) +

geom\_point(mapping = aes(x = theta3\_prior, y = theta\_bias\_prior)) +

ggtitle("Joint Density of Theta3 and Theta\_bias") +

xlab("Theta3") +

ylab("Theta\_bias") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

# Creating histogram of the marginal distribution of theta\_3

ggplot(data = data.frame(theta3 = accepted\_samples\_3)) +

geom\_histogram(mapping = aes(x = theta3), bins = 20, color = "white", fill = "black", alpha = 0.7) +

xlab("Theta3") +

ylab("Density") +

ggtitle("Histogram of Marginal Distribution of Theta3") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))

# Creating histogram of marginal distribution of theta\_bias

ggplot(data = data.frame(theta\_bias = accepted\_samples\_bias)) +

geom\_histogram(mapping = aes(x = theta\_bias), bins = 20, color = "white", fill = "black", alpha = 0.7) +

xlab("Theta\_bias") +

ylab("Density") +

ggtitle("Histogram of Marginal Distribution of Theta\_bias") +

theme\_bw() +

theme(plot.title = element\_text(hjust = 0.5, vjust = 0.5),

plot.margin = margin(10, 10, 10, 10, "mm"),

plot.background = element\_rect(fill = "white"))