An electroencephalogram (EEG) statistical analysis

Adu Boahene Quarshie
Faculty of Engineering, Environment and Computing
MSc. Data Science and Computational Intelligence
Coventry University, Coventry. United Kingdom
quarshiea@uni.coventry.ac.uk

Abstract: In this project we analyse an electroencephalogram dataset which we assume to be measured from two different regions of the brain. We then perform exploratory data analysis on the dataset and then move on to identify the non-linear regression model structure to estimate its parameters.

Keywords: EEG; Regression; R Code; Correlation;

I. INTRODUCTION

Here, we would like to model and understand the degree (or order) of possible nonlinear interactions between those two EEG channels. The basic model can be considered as a generic nonlinear polynomial regression model with the following (exemplar) structure:

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_n x^n + \epsilon$$

We then go ahead to perform exploratory data analysis on the dataset by analysing the time series plots of the signals, distribution of each signal, the signals correlation and fit a linear model to the data to estimate its performance. We will then identify the non-linear regression model structure and estimate its parameters by splitting the input and output signals into train and test sets. We then apply forward subset selection to get the best model structure. We also estimate parameters using the least mean squared error. When the best model is determined, we estimate the model's covariance matrix and compute the model's output or prediction. We finally validate the model and apply Approximate Bayesian Computation to compute the posterior distribution.

II. THE DATA SET

The dataset is an EEG time-series dataset with "X" and "Y" labels. It is assumed the signals are measured from two different regions of the brain. The output, Y, contains a corresponding additive noise with unknown variance. The dataset has 250 data points or entries.

III. EXPLORATORY DATA ANALYSIS

For us to be able to work with the data we have, we have to make sure that we achieve all the information needed to understand the nature of the data.

Renaming columns

When we load the data into R, we notice that the dataset is not labelled so R automatically column names V1 and V2. So, in order to make our dataset easy to work with, we do this with this R code:

```
#reading the csv file into R
data <- read.csv("x_y.csv", header = FALSE)
#renaming columns to input variable x and the output variable y
colnames(data)[colnames(data)=="V1"] <- "X"
colnames(data)[colnames(data)=="V2"] <- "y"
head(data)
```

Plotting time series for both input and output signals

A time series shows a progression of fluctuations of a particular variable in a time space, in this coursework we plot time series input against output to analyse how the pair moves together in that time space.

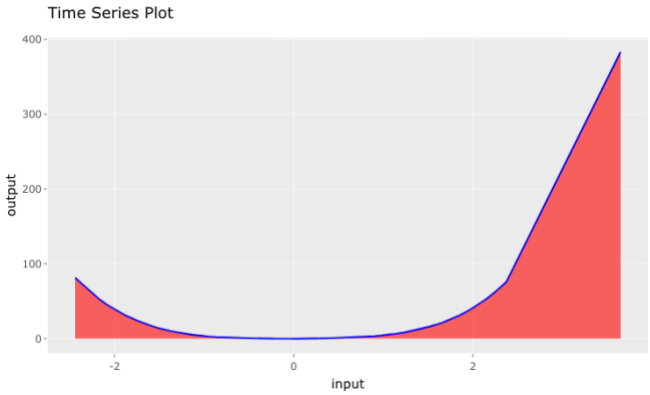

Fig. 1

The above plot shows the timeseries of both the input signal and its corresponding output signal. It is seen that when the input is ***-2.44104609***, the output signal is ***8.153268e^+01***.

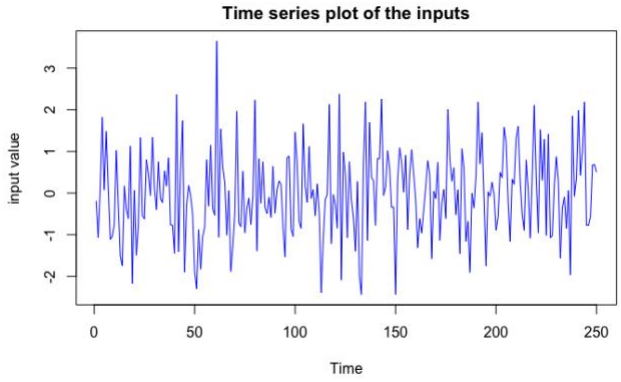

Fig. 2

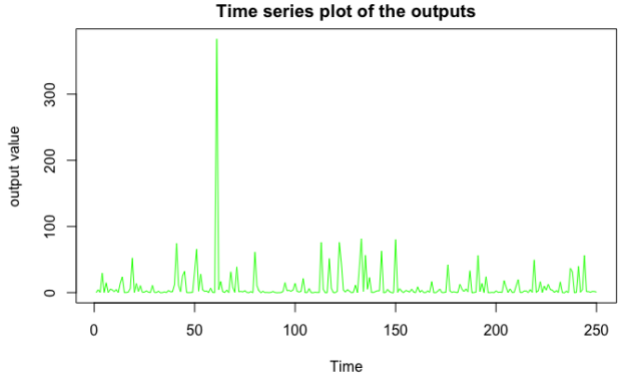

Fig 3

Plotting the individual time-series for both variables, the input signals has greater fluctuations as compared to the output signals. At about time, t = 60, the output signal is at its highest peak likewise the input signal.

Distribution for input and output signal

The plots below is the distribution of the Output signal variable. It can be seen that the output variable is highly skewed to the left.

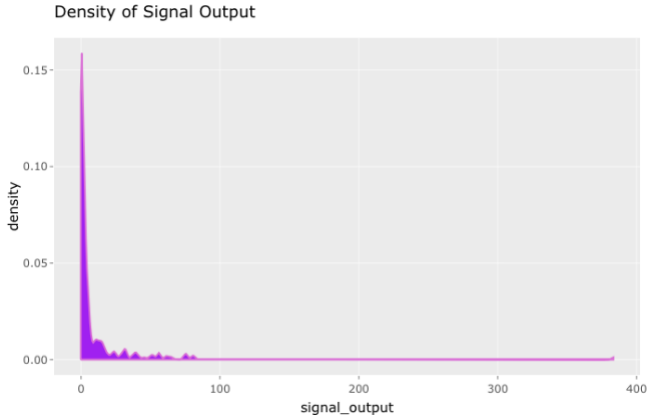

Fig 4.

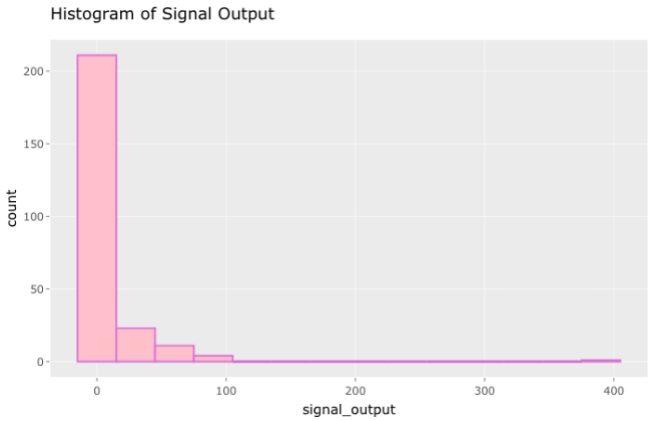

Fig 5.

As compared to the distribution of the output signal, the input signal is close to a normal distribution.

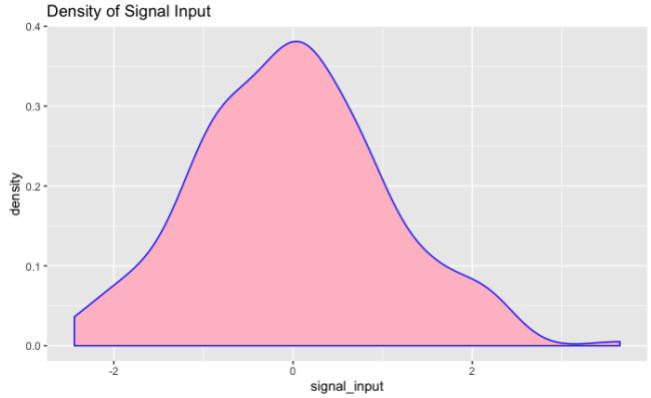

Fig 6.

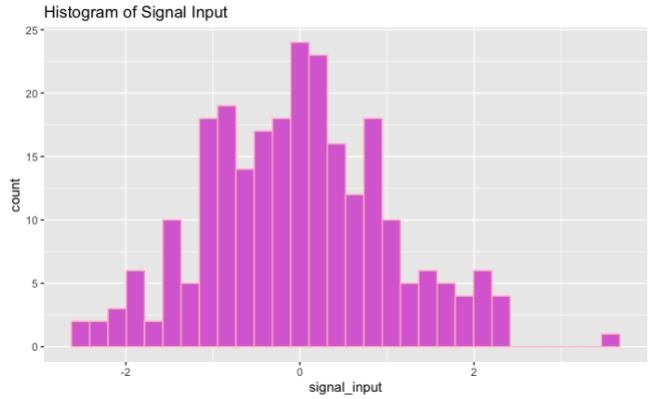

Fig 7.

Input and output correlation and scatter plot

It can be seen from the plot below that there exists an extreme outlier. This plot also shows the correlation between the signals.

For the correlation to be calculated, we use the Pearson's correlation formula.

Pearson correlation coefficient for population is given as;

$$\rho\, x, y = \frac{E[XY] - E[X]E[X]}{\sqrt{E[X^2] - [E[X]]^2}\,\sqrt{E[Y^2] - [E[Y]]^2}}$$

Pearson correlation coefficient for sample is given as;

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_1 - \bar{x})(y_1 - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_1 - \bar{x})^2}\,\sqrt{\sum_{i=1}^{n}(y_1 - \bar{y})^2}}$$

Using the 'cor()' function in R, it can be seen that there exists some sort of correlation between the input signal and the output signal from the diagram above. Performing the Pearson's correlation test, it is known that there exists a positive correlation of ***0.2193661*** between the input and output signals. R code below:

```
#correlation test
cor(data$X, data$y, method = "pearson", use = "complete.obs")
```

Model fitting and its performance

Fitting a static linear regression model to the data, thetaHat (the estimate of theta) was calculated to be ***5.844558***. And mean squared error (MSE) was ***886.2422***. This MSE shows the performance of the linear model fitted and since it is high, there is also a high residual value, therefore this model is not good. For the simple linear regression this is the formula;

$$y = \theta_0 + \theta_1 x + \epsilon$$

With θ_0 being the intercept and being zero computing the estimate for θ_1

IV. MODEL SELECTION

With the model selection, we create an intercept column, this creates a matrix of ones with 250 rows in one column. Then we create a matrix of 'X' that consist of the intercept and dependent variables of 'X'. We then split the dataset into test and train sets with 80% being for training and 20% being for testing.

Model selection (Forward model subset selection)

Forward selection is a kind of stepwise regression that starts with a null model and then adds variables in a succession. In each succession, a variable is added which makes our model better. For this part, we will create a data frame for Mean Squared Error, create a 'for' loop in R to find the minimum mean squared error values by using this code. We will then find the errors and append all the MSE's for each term in the data frame with the code below;

```
#Creating a data frame for MSE values
MSE_table = data.frame(terms =  colnames(X), MSE = rep(0, length(6)))

#Create a for loop to find the minimum MSE Values
```

```
for (i in 1:6){
  thetaHat = solve(t(X_train[,i]) %*% X_train[,i]) %*% t(X_train[,i]) %*% y_train

  y_hat = X_test[,i] %*% thetaHat

  #Finding the errors
  error = y_test - y_hat
  MSE = mean(error^2)
  MSE_table[i, 2] = MSE
}
```

Selecting minimum MSEs from data frame above

The X matrix has 250 values and I will be splitting it into a test and train set and choosing the ones with the minimum MSE from the MSE table computed below;

	Term	MSE
1	*Intercept*	177.950779
2	X	174.184985
3	X^2	93.988810
4	X^3	128.226234
5	X^4	5.232054
6	X^5	137.497411

Table 1.

Selecting the variables for the model which has the least MSE, from the first MSE table, it can be seen that X^4 had the least MSE as compared to the rest, therefore X^4 was selected and MSE was performed again on the rest of the variables, this was done repeatedly so as to select the variables with the least MSE, in order to create the best model.

Final Model

We then compute for the final model with the code below after getting all the least MSE's.

```
#Final model
xfinal_train = cbind(new_train3, new_train2, new_train1)
xfinal_test = cbind(new_test3, new_test2, new_test1)
colnames(xfinal_train) = c("X", "X^2", "X^4")
colnames(xfinal_test) = c("X", "X^2", "X^4")

final_thetaHat = solve(t(xfinal_train) %*% xfinal_train) %*% t(xfinal_train) %*% y_train

y_hat_final = xfinal_train %*% final_thetaHat #Refer to coursework guide about computing model prediction based on the train set

#Finding the errors
error = y_train - y_hat_final
MSE_final = mean(error^2)
```

The final model consists of X, X^2 and X^4. These variables/regressors give the least MSE.

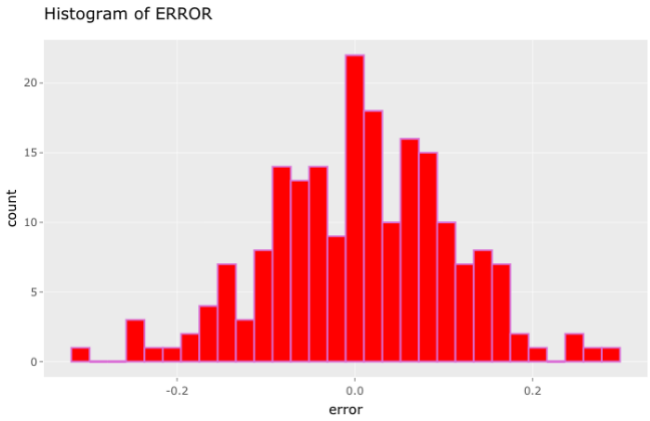

Fig 8.

Covariance

Covariance is the matrix that shows the variability between the variables. We calculate this in R with the code below;

```
error = y_train - y_hat_final
residuals_sum_of_squares = sum((error)^2)

sigma_square = residuals_sum_of_squares/n-1
#covariance matrix
cov_thetahat = sigma_square * (solve(t(xfinal_train) %*% xfinal_train)) #t(X) is transpose of X
colnames(cov_thetahat) = c("one_theta", "two_theta", "three_theta")
rownames(cov_thetahat) = c("one_theta", "two_theta", "three_theta")
```

Contour plotting and confidence interval

We plot contours for our combinations and then go ahead to calculate the confidence intervals. The code for the contour plotting and the 3D plots will be found in the appendix and the code for calculating the confidence interval is below;

```
n = 200
var_y_hat = matrix(0 , n , 1)

for( i in 1:n){
  X_i = matrix( xfinal_train[i,] , 1 , num_param ) # X[i,] creates a vector. Convert it to matrix
  var_y_hat[i,1] = X_i %*% cov_thetahat %*% t(X_i) # same as sigma_2 * ( X_i %*% ( solve(t(X) %*% X)  ) %*% t(X_i) )
}

CI = 2 * sqrt(var_y_hat) # Confidance interval

plot(data$X[1:200], y_hat_final, type = "p")
segments(data$X[1:200], y_hat_final-CI, data$X[1:200], y_hat_final+CI, col = "red") # Adds error bars to the indivigual data points
```

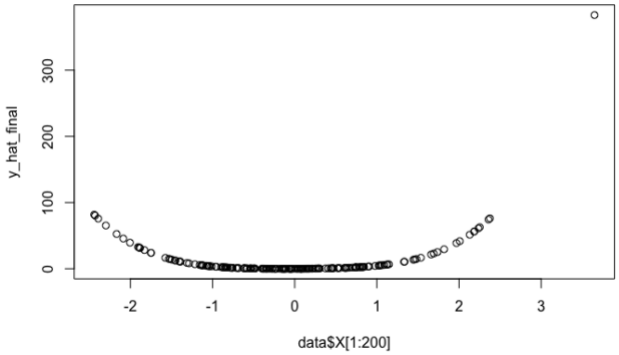

Fig 9.

IV. VALIDATION AND CONCLUSION

For validating our model, we will be splitting our dataset into 70% and 30% training and testing set respectively. We will then estimate the parameters for both test and train set and find the errors for both sets. The mean squared error is calculated to check the accuracy of the model.

From our computing with the code below, we find out that that the MSE_final = ***0.0109631630515089*** and the error rate of our test_MSE = ***0.00834867111029402***. This proves with the very less error that our model is the best fit for prediction for our dataset.

V. REFERENCES

[1] Brant, Rollin. Forward Selection. MDSC 643.02 Lecture Materials. Retrieved from https://www.stat.ubc.ca/~rollin/teach/643w04/lec/node41.html on July 7, 2018

[2] Cook, Perry. Stepwise Selection. Human-Computer Interface Technology (CS436) Class Notes. Retrieved from https://www.cs.princeton.edu/courses/archive/fall08/cos436/Duda/FS/stepwise.htm on July 8, 2018.

[3] Shalizi, Cosma. Lecture 26: Variable Selection. Modern Regression for Undergraduates Class Notes. http://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/26/lecture-26.pdf

[4] SAS Support. Forward Selection. The GLMSELECT Procedure. Retrieved from http://support.sas.com/documentation/cdl/en/statug/66859/HTML/default/viewer.htm#statug_glmselect_details03.htm on July 8, 2018

APPENDIX

Full Code and Plots for the coursework

EXPLORATORY DATA ANALYSIS

```
#reading the csv file into R
data <- read.csv("x_y.csv", header = FALSE)
#renaming columns to input variable x and the output variable y
colnames(data)[colnames(data)=="V1"] <- "X"
colnames(data)[colnames(data)=="V2"] <- "y"
head(data)
```

PLOTTING BOTH SIGNALS TIME-SERIES

```
library(plotly)
library(dplyr)
time_series <-
  ggplot(data = data, aes(x=data$X, y=data$y)) +
    geom_area(fill="red", alpha = 0.6) +
    geom_line(color="blue") +
  labs(x = "input", y = "output", title = "Time Series Plot")

# Turn it interactive with ggplotly
ggplotly(time_series)

plot.ts(data$X, col = "blue", main = "Time series plot of the inputs", ylab = "input value")
plot.ts(data$y, col = "green", main = "Time series plot of the outputs", ylab = "output value")
```

DISTRIBUTION FOR SIGNAL OUTPUT

```
library(ggplot2)
signal_output <- data$y
#plotting the density of the target variable
density <-ggplot(data = data, aes(signal_output)) +
  geom_density(kernel = "gaussian", col = "orchid", fill = "purple") +
  labs(title = "Density of Signal Output")
histogram <- ggplot(data = data, aes(signal_output)) +
  geom_histogram(col = "orchid", fill = "pink", binwidth = 30) +
  labs(title = "Histogram of Signal Output")
ggplotly(density)
ggplotly(histogram)
```

DISTRIBUTION FOR SIGNAL INPUT

```
signal_input <- data$X
#plotting the density of the input variable
ggplot(data = data, aes(signal_input)) +
  geom_density(kernel = "gaussian", col = "blue", fill = "pink") +
  labs(title = "Density of Signal Input")
ggplot(data = data, aes(signal_input)) +
  geom_histogram(col = "pink", fill = "orchid") +
  labs(title = "Histogram of Signal Input")
```

INPUT-OUTPUT CORRELATION AND TEST

```
ggplot(data, aes(x = data$X, y = data$y)) +
    geom_point(aes(color = "red")) +
  labs(x = "inputs" , y = "outputs")

#correlation test
cor(data$X, data$y, method = "pearson", use = "complete.obs")
```

MODEL FITTING AND ITS PERFORMANCE

```
thetaHat = solve(t(data$X)%*% data$X) %*% t(data$X) %*% data$y
print(thetaHat)
y_hat = data$X %*% thetaHat
Y_hat = data.frame(y_hat)
y_hat = Y_hat$y_hat
histogram_pred <- ggplot(data = Y_hat, aes(y_hat)) +
  geom_histogram(col = "orchid", fill = "pink") +
  labs(title = "Histogram of the predictions")
ggplotly(histogram_pred)

MSE = mean((data$y - y_hat)^2)
print (MSE)
error = data$y - y_hat
Error = data.frame(error)
error = Error$error
histogram_error <- ggplot(data = Error, aes(error)) +
  geom_histogram(col = "blue", fill = "red") +
  labs(title = "Histogram of the Error")
ggplotly(histogram_error)
```

MODEL SELECTION

```
#Creating an intercept column
#This creates a matrix of ones with 250 rows in one column
library(optimbase)
intercept = ones(nx = 250, ny = 1)

#Creating a matrix of X that consist of the intercept and dependent variables of X
X = data$X
X = cbind(intercept, X, X^2, X^3, X^4, X^5)
colnames(X) = c("intercept", "X", "X^2", "X^3", "X^4", "X^5")

#This changes y into a matrix
y <- data$y
y <- matrix(y)

#Splitting into train set and test set where 80% is for training and 20% is for testing
n = dim(X)[1]
#80% trainset
n_train = 0.8 * n
#splitting in train and test sets
X_train = X[1:n_train, ]
y_train = as.matrix(y[1:n_train, ])
X_test = X[(n_train+1):n, ]
y_test = as.matrix(y[(n_train+1):n, ])
```

FORWARD MODEL SUBSET SELECTION

```
#Creating a dataframe for MSE values
MSE_table = data.frame(terms =  colnames(X), MSE = rep(0, length(6)))

#Create a for loop to find the minimum MSE Values
for (i in 1:6){
  thetaHat = solve(t(X_train[,i]) %*% X_train[,i]) %*% t(X_train[,i]) %*% y_train
  y_hat = X_test[,i] %*% thetaHat

  #Finding the errors
  error = y_test - y_hati
  MSE = mean(error^2)
  MSE_table[i, 2] = MSE #Adding all the MSEs for each term in the dataframe
}
```

SELECTING THE MINIMUM MSE VALUE FROM THE DATA FRAME ABOVE

```
new_term1 = MSE_table$terms[which.min(MSE_table$MSE)]
```

```
#The X matrix has 250 values and I will be splitting it into a test and train set, and choosing th
e ones with the minimum MSE from the MSE table computed above.
new_train1 = as.matrix(X_train[, which.min(MSE_table$MSE)])
new_test1 = as.matrix(X_test[, which.min(MSE_table$MSE)])

#Removing X^4 from the modeland computing the MSE for the other variables in the model
X_train1 = X_train[, - which.min(MSE_table$MSE)]
X_test1 = X_test[, - which.min(MSE_table$MSE)]
MSE_table1 = data.frame(terms =  colnames(X_train1), MSE = rep(0, length(5))) #Creating a new data
frame

for (i in 1:5){
  train_x1 = cbind(new_train1, X_train1[, i])
  test_x1 = cbind(new_test1, X_test1[, i])
  thetaHat = solve(t(train_x1) %*% train_x1) %*% t(train_x1) %*% y_train

  y_hat = test_x1 %*% thetaHat

  #Finding the errors
  error = y_test - y_hat
  MSE1 = mean(error^2)
  MSE_table1[i, 2] = MSE1 #Adding all the MSEs for each term in the dataframe
  }

new_term2 = MSE_table1$terms[which.min(MSE_table1$MSE)]
#The X^2 has 250 values and I will be splitting it into a test and train set
new_train2 = as.matrix(X_train1[, which.min(MSE_table1$MSE)])
new_test2 = as.matrix(X_test1[, which.min(MSE_table1$MSE)])

#Creating a new X matrix without X^2
X_train2 = X_train1[, - which.min(MSE_table1$MSE)]
X_test2 = X_test1[, - which.min(MSE_table1$MSE)]
MSE_table2 = data.frame(terms =  colnames(X_train2), MSE = rep(0, length(4))) #Creating a new data
frame

for (i in 1:4){
  train_x2 = cbind(new_train2, new_train1, X_train2[, i])
  test_x2 = cbind(new_test2, new_test1, X_test2[, i])
  thetaHat = solve(t(train_x2) %*% train_x2) %*% t(train_x2) %*% y_train

  y_hat = test_x2 %*% thetaHat

  #Finding the errors
  error = y_test - y_hat
  MSE2 = mean(error^2)
  MSE_table2[i, 2] = MSE2 #Adding all the MSEs for each term in the dataframe
}

#New term 3
new_term3 = MSE_table2$terms[which.min(MSE_table2$MSE)]
#The X has 250 values and I will be splitting it into a test and train set
new_train3 = as.matrix(X_train2[, which.min(MSE_table2$MSE)])
new_test3 = as.matrix(X_test2[, which.min(MSE_table2$MSE)])
```

FINAL MODEL

```
#Final model
xfinal_train = cbind(new_train3, new_train2, new_train1)
xfinal_test = cbind(new_test3, new_test2, new_test1)
colnames(xfinal_train) = c("X", "X^2", "X^4")
colnames(xfinal_test) = c("X", "X^2", "X^4")

final_thetaHat = solve(t(xfinal_train) %*% xfinal_train) %*% t(xfinal_train) %*% y_train

y_hat_final = xfinal_train %*% final_thetaHat #Refer to coursework guide about computing model pre
diction based on the train set

#Finding the errors
error = y_train - y_hat_final
MSE_final = mean(error^2)

```

```
 Error = data.frame(error)
 error = Error$error
histogram_error <- ggplot(data = Error, aes(error)) +
  geom_histogram(col = "orchid", fill = "red") +
  labs(title = "Histogram of ERROR ")
ggplotly(histogram_error)
```

COVARIANCE

```
error = y_train - y_hat_final
residuals_sum_of_squares = sum((error)^2)
sigma_square = residuals_sum_of_squares/n-1
#covariance matrix
cov_thetahat = sigma_square * (solve(t(xfinal_train) %*% xfinal_train)) #t(X) is transpose of X
colnames(cov_thetahat) = c("one_theta", "two_theta", "three_theta")
rownames(cov_thetahat) = c("one_theta", "two_theta", "three_theta")
```

CONTOUR PLOTTING

```
num_param = 3
num_points = 50 # no point on the plot
one_theta = seq(0.489 , 0.509 , length=num_points)
two_theta = seq(1.993 , 2.013 , length=num_points)
three_theta = seq(1.990 , 2.010 , length=num_points)

prob_den_func = matrix(0 , num_points , num_points)

cov_thetahat_inv = (t(X) %*% X) * (1/sigma_square) # inverse of cov_thetaHat
det_cov_thetahat = det(cov_thetahat) # determinent of cov_thetaHat

#theta_1 and theta_2 combination
for(r in 1:50){
  for(c in 1:50){

    one_two_theta = matrix( c(one_theta[r] , two_theta[c], final_thetaHat[3,] ) , num_param , 1)
    thetahat_theta = one_two_theta - final_thetaHat

    prob_den_func[r,c] = ( 1/sqrt( ( (2*pi)^num_param ) * det_cov_thetahat) ) *
     exp( -0.5 * t(-thetahat_theta) %*% solve(cov_thetahat) %*% -thetahat_theta )

  }
}

contour(one_theta, two_theta, prob_den_func)
persp(one_theta, two_theta, prob_den_func, theta = 50 , phi = 50)

#one_theta and three_theta combination

for(r in 1:50){
  for(c in 1:50){

    one_three_theta = matrix( c(one_theta[r], final_thetaHat[2,] , three_theta[c] ) , num_param, 1
)
    thetahat_theta = one_three_theta - final_thetaHat

    prob_den_func[r,c] = ( 1/sqrt( ( (2*pi)^num_param ) * det_cov_thetahat) ) *
     exp( -0.5 * t(-thetahat_theta) %*% solve(cov_thetahat) %*% -thetahat_theta )

  }
}

contour(one_theta, three_theta, prob_den_func)
persp(one_theta, three_theta, prob_den_func, theta = 50 , phi = 50)


#two_theta and three_theta combination

for(r in 1:50){
  for(c in 1:50){
```

```
50. 
51.     two_three_theta= matrix( c(final_thetaHat[1,], two_theta[r] , three_theta[c] ) , num_param , 1
    )
52.     thetahat_theta = two_three_theta - final_thetaHat
53. 
54.     prob_den_func[r,c] = ( 1/sqrt( ( (2*pi)^num_param ) * det_cov_thetahat) ) *
55.     exp( -0.5 * t(-thetahat_theta) %*% solve(cov_thetahat) %*% -thetahat_theta )
56. 
57.   }
58. }
59. 
60. contour(two_theta, three_theta, prob_den_func)
61. persp(two_theta, three_theta, prob_den_func, theta = 50 , phi = 50)
62. ```
63. 
64. #Confidence Interval
65. ```{r}
66. n = 200
67. var_y_hat = matrix(0 , n , 1)
68. 
69. for( i in 1:n){
70.   X_i = matrix( xfinal_train[i,] , 1 , num_param ) # X[i,] creates a vector. Convert it to matrix

71.   var_y_hat[i,1] = X_i %*% cov_thetahat %*% t(X_i) # same as sigma_2 * ( X_i %*% ( solve(t(X) %*%
    X)  ) %*% t(X_i) )
72. }
73. 
74. CI = 2 * sqrt(var_y_hat) # Confidance interval
75. 
76. plot(data$X[1:200], y_hat_final, type = "p")
77. segments(data$X[1:200], y_hat_final-
    CI, data$X[1:200], y_hat_final+CI, col = "red") # Adds error bars to the indivigual data points
```

CONFIDENCE INTERVAL

```
1.  n = 200
2.  var_y_hat = matrix(0 , n , 1)
3.  
4.  for( i in 1:n){
5.    X_i = matrix( xfinal_train[i,] , 1 , num_param ) # X[i,] creates a vector. Convert it to matrix

6.    var_y_hat[i,1] = X_i %*% cov_thetahat %*% t(X_i) # same as sigma_2 * ( X_i %*% ( solve(t(X) %*%
    X)  ) %*% t(X_i) )
7.  }
8.  
9.  CI = 2 * sqrt(var_y_hat) # Confidance interval
10. 
11. plot(data$X[1:200], y_hat_final, type = "p")
12. segments(data$X[1:200], y_hat_final-
    CI, data$X[1:200], y_hat_final+CI, col = "red") # Adds error bars to the indivigual data points
```

VALIDATION

```
1.  X_data_matrix = X[, c(2, 3, 5)]
2.  colnames(X_data_matrix) = c("X","X^2", "X^4")
3.  
4.  #Splitting the data into 70/30 sets
5.  x_train_val = X_data_matrix[1:175,]
6.  y_train_val = as.matrix(data$y[1:175])
7.  x_test_val = X_data_matrix[176:250,]
8.  y_test_val = as.matrix(data$y[176:250])
9.  
10. 
11. #Estimation of Parameters
12. val_thetaHat = solve(t(x_train_val) %*% x_train_val) %*% t(x_train_val) %*% y_train_val
13. 
14. y_hat_test = x_test_val %*% val_thetaHat #for testing
15. y_hat_train = x_train_val %*% val_thetaHat #for training
16. 
```

```
#Finding error
test_error = y_test_val - y_hat_test #Testing error
train_error = y_train_val - y_hat_train #Training error

#MSE
test_MSE = mean((test_error)^2)
train_MSE = mean((train_error)^2)

#Comment on the above to prove the validity of the model.
```
 Validation is checking if the chosen model is the best fit or accurate for the dataset
 Compared the MSE

 #ABC
```{r include=FALSE}
pacman::p_load(ggplot2, ggthemes, tidyr, patchwork, plotly, viridis)
```

PRIORS

```
sampling_n = 4e7 #draw 30M from each prior

prior_x1 = runif(sampling_n, -5, 5)
prior_x2 = runif(sampling_n, -5, 5)
prior_x4 = runif(sampling_n, -5, 5)

#merge priors to one matrix so we can simply take rows as theta candidates
priors = cbind(prior_x1, prior_x2, prior_x4)
priors = as.data.frame(priors)
colnames(priors) = c("x1", "x2", "x4")

rm(prior_x1)
rm(prior_x2)
rm(prior_x4)
```

PLOTTING THE PRIORS

```
priors_long = gather(priors, prior, sample, x1:x4, factor_key = T)

(priors_plot = ggplot(priors_long, aes(x = sample, color = prior))+
  geom_density()+
  facet_wrap(~prior, scales = "free")+
  theme_few()+
  scale_color_few()+
  guides(color = F)+
  xlab(""))

ggsave("figures/10_priors.png", width = 7, height = 4)
rm(priors_long)
rm(priors_plot)

X = cbind(data$x,
          data$x^2,
          data$x^4
          )
```

REJECTION ABC

```
posterior =data.frame(x1 = numeric(),
                      x2 = numeric(),
                      x4 = numeric())

#cross validation testing MSE +3*sd
#tolerance = 0.028

tolerance = 0.2


```

```
for (i in 1:nrow(priors)) {
  candidate = t(as.matrix(priors[i,]))
  simulated = X %*% candidate

  MSE = mean((data$y - simulated)^2)

  if (MSE < tolerance) {
    append = t(candidate)
    colnames(append) = c("x1", "x2", "x4")
    posterior = rbind(posterior, append)
  }
}

cat(nrow(posterior), "samples were accepted.")

write.csv(posterior, "data/posterior.csv")



h1 = ggplot(posterior, aes(x = x1))+
  geom_density()+
  labs(x = "", y="")+
  theme_few()
h2 = ggplot(posterior, aes(x = x2))+
  geom_density()+
  labs(x = "", y="")+
  theme_few()
h3 = ggplot(posterior, aes(x = x4))+
  geom_density()+
  labs(x = "", y="")+
  scale_x_continuous(expand = c(0, 0),
                     breaks = round(seq(min(posterior$x4), max(posterior$x4), by = 0.02),2))+
  theme_few()

p1 = ggplot(posterior, aes(x=x1, y=x2) ) +
  stat_density_2d(aes(fill = ..density..), geom = "raster", contour = FALSE) +
  scale_fill_viridis() +
  scale_x_continuous(expand = c(0, 0),
                     breaks = round(seq(min(posterior$x1), max(posterior$x1), by = 0.2),1)) +
  scale_y_continuous(expand = c(0, 0),
                     breaks = round(seq(min(posterior$x2), max(posterior$x2), by = 0.2),1)) +
  theme(
    legend.position='none'
  )+
  guides(fill = F)+
  labs(x = "", y="")+
  theme_few()

p2 = ggplot(posterior, aes(x=x1, y=x4) ) +
  stat_density_2d(aes(fill = ..density..), geom = "raster", contour = FALSE) +
  scale_fill_viridis() +
  scale_x_continuous(expand = c(0, 0),
                     breaks = round(seq(min(posterior$x1), max(posterior$x1), by = 0.2),1)) +
  scale_y_continuous(expand = c(0, 0),
                     breaks = round(seq(min(posterior$x4), max(posterior$x4), by = 0.05),2)) +
  theme(
    legend.position='none'
  )+
  guides(fill = F)+
  labs(x = "", y="")+
  theme_few()

p3 = ggplot(posterior, aes(x=x2, y=x4) ) +
  stat_density_2d(aes(fill = ..density..), geom = "raster", contour = FALSE) +
  scale_fill_viridis() +
  scale_x_continuous(expand = c(0, 0),breaks = round(seq(min(posterior$x2), max(posterior$x2), by
= 0.2),1)) +
  scale_y_continuous(expand = c(0, 0),
                     breaks = round(seq(min(posterior$x4), max(posterior$x4), by = 0.05),2)) +
  theme(
    legend.position='none'
  )+
  guides(fill = F)+
```

```
  labs(x = "", y="")+
  theme_few()

final_plot =
  (h1+(p1+coord_flip())+(p2+coord_flip()))/
  (p1+h2+(p3+coord_flip()))/
  (p2+p3+h3)
ggsave("figures/11_ABC_posterior.png", width = 7, height = 4)
```

GETTING THE MAP

```
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

cat("x: ",getmode(posterior$x1))
cat("\nx2:", getmode(posterior$x2))
cat("\nx4:", getmode(posterior$x4))
```

PREDICTION GENERATION

```
posterior = read.csv("data/posterior.csv", row.names = 1)
predictions = data.frame()

for (i in 1:nrow(posterior)) {
    candidate = t(as.matrix(posterior[i,]))
    simulated = X %*% candidate
    append = data.frame(y = simulated, which = rep(i, length(simulated)))
    colnames(append)[1] = "y"
    predictions = rbind(predictions, append)
}

predictions$which = as.factor(predictions$which)
predictions$x = rep(data$x, nrow(posterior))
predictions$truth = rep(data$y, nrow(posterior))

(plot_predictions = ggplot(predictions, aes(x = x, y = y, color=which))+
  geom_line(alpha = 0.2)+
  geom_line(aes(y = truth), color="black")+
  guides(color = F)+
  theme_few())

ggsave("figures/12_bayesian_prediction.png", plot_predictions, width = 7, height = 4)
```

CONTOUR PLOTINGS