Pendulum Factorial Design Experiment

Introduction

In this experiment, we wish to study the period of a pendulum; the period being the amount of time required for the pendulum to complete a full cycle. The pendulum consists of 2 main components: the weight/mass, and the element used to suspend this weight at a fixed point while allowing the weight to swing freely (i.e. a piece of string or a rod). Furthermore, the idea for this experiment was came from my interest in how the piano metronome keeps its beat depending how it is adjusted.

As this is a factorial experiment, we will study the effects of the pendulum's mass, string length, and amplitude (also known as the drop angle of the pendulum) on its period. The 3 factors will each consist of 2 levels. The mass factor has a high mass level (300g) and low mass level (100g). The string length has a long length level (50cm) and a short length level (10cm). Lastly, the amplitude will have a high amplitude level (65^o) and a low amplitude level (15^o). Given this, we are conducting a full factorial experiment with 2^3 unique combinations. To reduce the amount of variability in the study, the 8 combinations were replicated once more to provide more reliable data.

Procedure and Data

In this experiment, one end of a string (length depends on the level we wish to observe) was taped down to the edge of a table. On the other end of the string, a container was attached such that the container could hold a variety of masses. It should be noted that the total masses were measured using a scale prior to being attached onto the pendulum.

To start the first run of the experiment, a 100g mass was attached to a 10cm string. Next, with the use of a protractor, the weight was brought to an amplitude of 15^o and released. As soon as the weight was dropped, a stopwatch was started to record the time taken to complete 10 full cycles (1 cycle = a full back and forth swing) and then recorded onto an excel spreadsheet. After the first run, we ran the experiment 7 more times while varying the string length, mass and amplitude each time.

The whole procedure above was then replicated once more to create another 8 runs. Below, shows my general setup of the experiment, and the data collected on the excel spreadsheet.

Note: The data from the first run/trial are Samples 1,3,5,7,…,15

Sample	Period (Sec)	String Length (Cm)	Mass (g)	Amplitude (degrees)
1	6.36	10	100	15
2	6.2	10	100	15
3	14.29	50	100	15
4	14.23	50	100	15
5	6.49	10	300	15
6	6.4	10	300	15
7	14.26	50	300	15
8	14.3	50	300	15
9	7	10	100	65
10	6.86	10	100	65
11	15.47	50	100	65
12	15.51	50	100	65
13	6.99	10	300	65
14	6.89	10	300	65
15	15.5	50	300	65
16	15.3	50	300	65

Results

Prior to making any further analyses of the data, the average period was calculated between the 2 experimental runs and the resulting table can be seen in the *Appendix* under the name 'avgData'. To analyse the data, functions were created on R (see *Appendix*) called 'mainEffectSL', 'mainEffectMas', 'mainEffectAmp'; these functions were used to calculate the average main effect for each factor in the experiment. Furthermore, similar functions were used to calculate the three 2-factor interaction effects, and one 3-factor interaction effect. Below is a table with the calculated main effects, interactions effects, and their respective 95% confidence interval (it should be mentioned that the confidence intervals are calculated using a pooled variance value of 0.00688; this can be found under the function 'pooledVar'):

Main & Interaction Effects	Values	2.5% Lower Limit	97.5 Upper Limit
String Length	8.209	8.113	8.304
Mass	0.026	-0.069	0.121
Amplitude	0.874	0.778	0.969
String Length: Mass	0.0612	-0.034	0.156
String Length: Amplitude	0.3012	0.205	0.396
Amplitude: Mass	0.0662	-0.029	0.161
Amplitude: Mass: String Length	0.0112	-0.1240	0.146

Note: All calculations and functions are shown in the Appendix

In addition, a linear model was fitted onto the full set of data points collected. This linear model can be viewed in the *Appendix* under the variable name 'linModel'. Using this linear model, we have also created a half-normal plot to further observe which factors were most significant in this study. Furthermore, we can also describe the linear model by the following equation below:

$Y_i = \beta_0 + \beta_1 S + \beta_2 M + \beta_3 A + \beta_4 SM + \beta_5 SA + \beta_6 AM + \beta_7 SMA + \varepsilon_i$ where i represents the i^{th} run

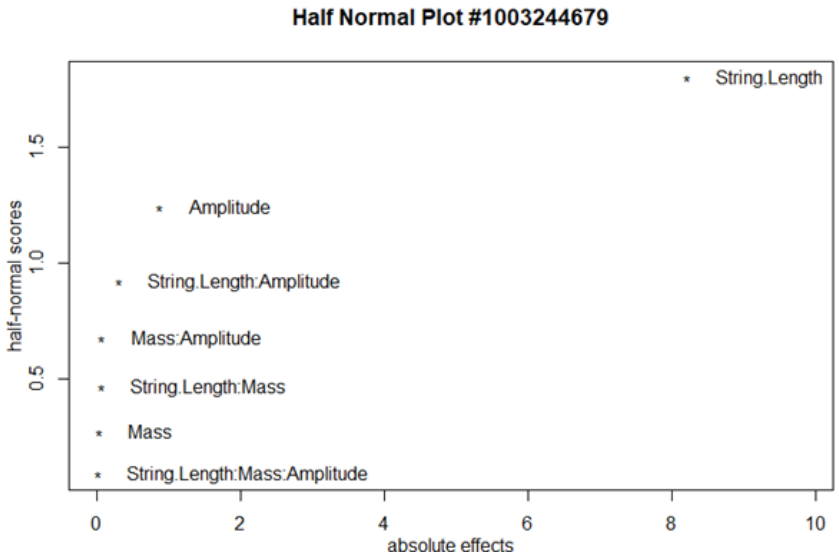
- Y_i represents the period of the pendulum for the i^{th} experimental run
- S represents the pendulum's String Length factor; with levels -1 = 10 cm and 1 = 50cm
- M represents the pendulum's Mass factor; with levels -1 = 100g and 1 = 300g
- A represents the pendulum's Amplitude; with levels -1 = 15 degrees and 1 = 65 degrees
- ε_i is the error

Below shows a table of the estimated coefficient values for the linear model:

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              10.753125   0.020738 518.515  < 2e-16 ***
String.Length             4.104375   0.020738 197.913 4.75e-16 ***
Mass                      0.013125   0.020738   0.633    0.544
Amplitude                 0.436875   0.020738  21.066 2.71e-08 ***
String.Length:Mass       -0.030625   0.020738  -1.477    0.178
String.Length:Amplitude   0.150625   0.020738   7.263 8.69e-05 ***
Mass:Amplitude           -0.033125   0.020738  -1.597    0.149
String.Length:Mass:Amplitude  0.005625   0.020738   0.271    0.793
```

Upon viewing the 2 tables above, it is clear that both methods seem to suggest that a pendulum's string length, amplitude, and their interaction has a significant effect on the period of the pendulum. In the first table, we see that the factors which do not include 0 are: string length, amplitude, and the string length & amplitude interaction. Similarly, in our linear model, the coefficients which have significant p-values (i.e. less that our significance level of 5%) are the same as the factors in the first table.

To further support the significance of these 3 effects, the half-normal plot below indicates that string length has the highest effect on the period. After string length, we see that amplitude has the second highest effect. And lastly, the interaction effect between the string length and the amplitude appears to have the least significant effect on the period; however, it is still more significant then the mass; mass & amplitude interaction; string length & mass interaction; and the 3-factor interaction effect.



Conclusion

In conclusion, from our previous plot and tables, we have strong evidence to conclude that a pendulum's period is primarily affected by its string length. In other words, a pendulum which is suspended by a longer string will have a larger increase in its period in comparison to a shorter string. Secondly, we have also observed that a pendulum's amplitude can also have a significant effect on the period. By increasing the amplitude of the pendulum, we have seen that the period also increases; however, it is worth noting that the increase in the period is not as large here as it is in the previous case. Furthermore, if both the string length and amplitude are increased, we will see relatively small increases to the period (as seen from the calculated values and coefficients for the "String.Length: Amplitude" interaction effect in contrast to the main effects of string length and amplitude). Lastly, of the 3 factors which were analysed, only 1 of 3 factors were not significant. The mass of the pendulum appears to have no significant effect on the period, thus, the increase or decrease of the weight will not result in a big change to the period's time.

Appendix

Dataset

Below is a table which contains the average period values between the main and replicated trials from the *Procedures and Data* section:

avgData <- swingData[1:8, 8:12]

```
> avgData
  Sample Avg Period String.Length.1 Mass.1 Amplitude.1
1      1      6.280               10    100          15
2      2     14.260               50    100          15
3      3      6.445               10    300          15
4      4     14.280               50    300          15
5      5      6.930               10    100          65
6      6     15.490               50    100          65
7      7      6.940               10    300          65
8      8     15.400               50    300          65
```

Main Effect Functions and Calculations

(Note: 'avgData' has been set to be the default for **all main & interaction** effects functions)

Below are the created functions which loops through the period of our 'avgData' to determine its *main effects vector*. The functions essentially take the period difference between the desired factor at its +1 level and -1 level.

There are 3 different functions below where the short forms 'SL', 'Mas', 'Amp' respectively represents string length, mass, and amplitude. For a clearer understanding of the functions made, a more explicit sample calculation for the first function 'mainEffectSL' is shown below:

$Period_2 - Period_1 = 14.260 - 6.280 = 7.980$
$Period_4 - Period_3 = 14.280 - 6.445 = 7.835$
$Period_6 - Period_5 = 15.490 - 6.930 = 8.560$
$Period_8 - Period_7 = 15.400 - 6.940 = 8.460$

emptyVector = [7.980, 7.835, 8.560, 8.460] <- Here is a vector of the differences in avg periods. To see the final *average* main effect, please look for (**) near the end of this section

```
mainEffectSL <- function(dataTable = avgData){
  emptyEffect <- numeric()
  for (i in c(2,4,6,8)){
    effect <- dataTable[i,2] - dataTable[i-1, 2]
    emptyEffect <- append(emptyEffect, effect)
          }
  return(emptyEffect)
}

mainEffectMas <- function(dataTable = avgData){
  emptyEffect <- numeric()
  for (i in c(3,4,7,8)){
    effect <- dataTable[i,2] - dataTable[i-2, 2]
    emptyEffect <- append(emptyEffect, effect)
          }
```

```
  return(emptyEffect)
}

mainEffectAmp <- function(dataTable = avgData){
  emptyEffect <- numeric()
  for (i in c(5,6,7,8)){
    effect <- dataTable[i,2] - dataTable[i-4, 2]
    emptyEffect <- append(emptyEffect, effect)
          }
  return(emptyEffect)
}
```

******Below shows the main average effects using the *main effects vector* from the previous functions

```
avgMainEffSL <- mean(mainEffectSL())
avgMainEffMA <- mean(mainEffectMas())
avgMainEffAM <- mean(mainEffectAmp())
```

Interaction Effect Function and Calculations

Below shows the functions used to calculate the 2-factor and 3-factor interaction effect values. The function creates an empty vector, 'emptyEffect', and calculates 2 averages depending on the desired interaction. Then the difference is taken between the 2 previously calculated averages and then averaged.
Once again, a more explicit calculation will be shown below for the function 'intEffectSLAmp':

$(Period_2 + Period_4) /2 – (Period_1 + Period_3) /2 = (14.260 + 14.280) /2 – (6.280 + 6.445) /2 = 7.907$
$(Period_6 + Period_8) /2 – (Period_5 + Period_7) /2= (15.490 + 15.400) /2 – (6.930 + 6.940) /2 = 8.510$

Interaction effect of SL and Amp= (8.510 – 7.907)/2 = 0.301

```
intEffectSLAmp <- function(dataTable = avgData){
  emptyEffect <- numeric()
  for (i in c(1,5)){
    effect <- (dataTable[i+1,2] + dataTable[i+3, 2])/2 - (dataTable[i,2] + dataTable[i+2, 2])/2
    emptyEffect <- append(emptyEffect, effect)
           }
  emptyEffect <- (emptyEffect[2] - emptyEffect[1])/2
  return(emptyEffect)
}


intEffectSLMass <- function(dataTable = avgData){
  emptyEffect <- numeric()
  for (i in c(1,3)){
    effect <- (dataTable[i+5, 2] + dataTable[i + 1, 2])/2 - (dataTable[i + 4, 2] + dataTable[i, 2])/2
    emptyEffect <- append(emptyEffect, effect)
          }
  emptyEffect <- (emptyEffect[1] - emptyEffect[2]) / 2
  return(emptyEffect)
}
```

```
intEffectMassAmp <- function(dataTable = avgData){
  emptyEffect <- numeric()
  for (i in c(1,3)){
    effect <- (dataTable[i+5, 2] + dataTable[i + 4, 2])/2 - (dataTable[i, 2] + dataTable[i + 1, 2])/2
    emptyEffect <- append(emptyEffect, effect)
          }
  emptyEffect <- (emptyEffect[1] - emptyEffect[2])/2
  return(emptyEffect)
}

tripleEff <- function(dataTable=avgData){
  emptyEffect <- numeric()
  for (i in c(8,4)){
    effect <- ((dataTable[i, 2] - dataTable[i-1,2]) - (dataTable[i-2, 2] - dataTable[i-3, 2])) / 2
    emptyEffect <- append(emptyEffect, effect)
          }
  tripleInt <- (emptyEffect[1] - emptyEffect[2]) / 2
  return(tripleInt)
}
```

Pooled Variance Calculations

Below shows a function that utilizes the **full dataset** (as seen from the Procedures & Data section) to calculate the pooled variance. This function creates an empty vector called 'emptyDiff' that will be filled with the differences between the first run's and the replicated run's period. When 'emptyDiff' is filled, the pooled variance, 'pooledVariance', is found using this vector with the operations shown below.

```
pooledVar <- function(dataTable = swingDataFull){
  emptyDiff <- numeric()
  for (i in seq(1,16,2)){
    difference <- dataTable[i+1, 2] - dataTable[i, 2]
    emptyDiff <- append(emptyDiff,difference)
          }
  pooledVariance <- sum((emptyDiff)^2) / (8 * 2)
  return(pooledVariance)
}
```

Standard Error and Confidence interval Calculations

The standard error was first computed by taking the square-root of the previously calculated pooled variance. Using this standard error with the test statistic variable, we calculated the 95% confidence intervals for main effects, 2-factor interaction and 3-factor interaction effects.

```
standardError <- sqrt((1/8 + 1/8) * pooledVar())
testStat <- qt(0.025, 8, lower.tail = FALSE)

cIntervalM1 <- c(avgMainEffSL - testStat*standardError, avgMainEffSL + testStat*standardError)
cIntervalM2 <- c(avgMainEffMA - testStat*standardError, avgMainEffMA + testStat*standardError)
```

cIntervalM3 <- c(avgMainEffAM - testStat*standardError, avgMainEffAM + testStat*standardError)

cIntervalI1 <- c(intEffectSLAmp() - testStat*standardError, intEffectSLAmp() + testStat*standardError)
cIntervalI2 <- c(intEffectMassAmp() - testStat*standardError, intEffectMassAmp() + testStat*standardError)
cIntervalI3 <- c(intEffectSLMass() - testStat*standardError, intEffectSLMass() + testStat*standardError)

cIntervalI4 <- c(tripleEff() - testStat*standardError, tripleEff() + testStat*standardError)

Linear Modeling & Half-normal Plot

```
linModel <- lm(Period~ String.Length * Mass * Amplitude, data = numCodedData)
DanielPlot(linModel, half = TRUE, autolab = F, main= "Half Normal Plot #1003244679")
```