SMDE: Homework 2

Arnau Abella
Universitat Politècnica de Catalunya

May 26, 2020

1 Simulate your data

The choosen answer variable is:

$$Answer = f_1 + f_2 + f_4 + 5f_5 \tag{1}$$

The generated system can be found at `ex1/dataset.csv`.

In order to generate a new dataset, you only need to run the script `./ex1/Main.hs` that I specifically prepared to simplify this task. Before that, you need to install Nix which will bring all the dependencies to your environment to be able to run the script.

In the following pages you will find the source code of this script 1. The script was written in *Haskell* but it should be easy to follow (modulo some language-specific details).

```
#!/usr/bin/env nix-shell
#!nix-shell -i runghc
{-# LANGUAGE DeriveAnyClass     #-}
{-# LANGUAGE DeriveGeneric      #-}
{-# LANGUAGE DerivingStrategies #-}
module Main where

import            Data.ByteString.Builder
import            Data.ByteString.Lazy
import            Data.Csv
import            GHC.Generics
import            Prelude                           hiding (writeFile)
import            System.Random.MWC
import            System.Random.MWC.Distributions

data Individual = Individual
  { factor1  :: Double
  , factor2  :: Double
  , factor3  :: Double
  , factor4  :: Double
  , factor5  :: Double
  , factor6  :: Double
  , factor7  :: Double
  , factor8  :: Double
  , factor9  :: Double
  , factor10 :: Double
  , answer   :: Double
  } deriving stock (Show, Generic)
    deriving anyclass (ToRecord, ToNamedRecord, DefaultOrdered)

generateIndividual :: IO Individual
generateIndividual = withSystemRandom . asGenIO $ \gen -> do
  f1 <- abs <$> standard gen
  f2 <- abs <$> uniform gen
  f3 <- abs <$> exponential 0.5 gen
  f4 <- abs <$> standard gen
  f5 <- abs <$> uniform gen
  let f6 = f1 + f2
      f7 = f1 + 2*f3
      f8 = f1 + f4
      f9 = f4 + 5*f5
```

```
    f10 = f1 + f2 + f3 + f4 + f5
  answer <- (f1 + f2 + f9 +) <$> standard gen
  return (Individual f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 answer)

generateDataset :: Int -> IO [Individual]
generateDataset 0 = return []
generateDataset n = do
  x  <- generateIndividual
  xs <- generateDataset (n - 1)
  return (x : xs)

main :: IO ()
main =
  generateDataset 2000
    >>= writeFile "dataset.csv" . encodeDefaultOrderedByName
```

2 Obtain an expression to generate new data

The following two techniques have been used for exploring the dataset's interactions of the different factors:

- Multiple Linear Regression Model.
- Principal Component Analysis.

The resulting expression from the *LRM* is

$$Answer' = 0.005562 + 1.032f_1 + 0.988f_2 + 0.988f_4 + 4.94f_5 \tag{2}$$

which is a very good approximation of the system answer as we will see later.

In the following pages include the scripts, observations and validations of the dataset and its resulting modelling expression.

2.1 Multiple Linear Regression Model

Let's start by applying a *multiple linear regression model* to the generated dataset from the first exercise:

```
reg_model1<-lm(answer~., data=dataset)
summary(reg_model1)
```

```
##
## Call:
## lm(formula = answer ~ ., data = dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1928 -0.7032 -0.0417  0.6934  3.2072
##
## Coefficients: (5 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.005562   0.077038   0.072    0.942
## factor1      1.031851   0.038688  26.671   <2e-16 ***
## factor2      0.987810   0.079949  12.355   <2e-16 ***
## factor3     -0.008692   0.011644  -0.746    0.455
## factor4      0.988821   0.037947  26.058   <2e-16 ***
## factor5      4.942126   0.079995  61.780   <2e-16 ***
## factor6            NA         NA      NA       NA
## factor7            NA         NA      NA       NA
## factor8            NA         NA      NA       NA
## factor9            NA         NA      NA       NA
## factor10           NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.02 on 1994 degrees of freedom
## Multiple R-squared:  0.7327, Adjusted R-squared:  0.732
## F-statistic:  1093 on 5 and 1994 DF,  p-value: < 2.2e-16
```

The implementation of `lm` is clever enough to detect that the factors f6-f10 are a linear combination of the factors f1-5.

We can analyze them separately. As expected, only factor 6 and factor 9 have a linear relation with the answer.

```
reg_model2<-lm(answer~factor6+factor7+factor8+factor9+factor10, data=dataset)
summary(reg_model2)
```

```
##
## Call:
## lm(formula = answer ~ factor6 + factor7 + factor8 + factor9 +
##     factor10, data = dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1928 -0.7032 -0.0417  0.6934  3.2072
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.005562   0.077038   0.072    0.942
## factor6      1.024726   0.047754  21.459   <2e-16 ***
```

```
## factor7       0.014112   0.038168   0.370    0.712
## factor8       0.029929   0.057247   0.523    0.601
## factor9       0.995808   0.023845  41.762   <2e-16 ***
## factor10     -0.036916   0.075801  -0.487    0.626
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.02 on 1994 degrees of freedom
## Multiple R-squared:  0.7327, Adjusted R-squared:  0.732
## F-statistic:  1093 on 5 and 1994 DF,  p-value: < 2.2e-16
```

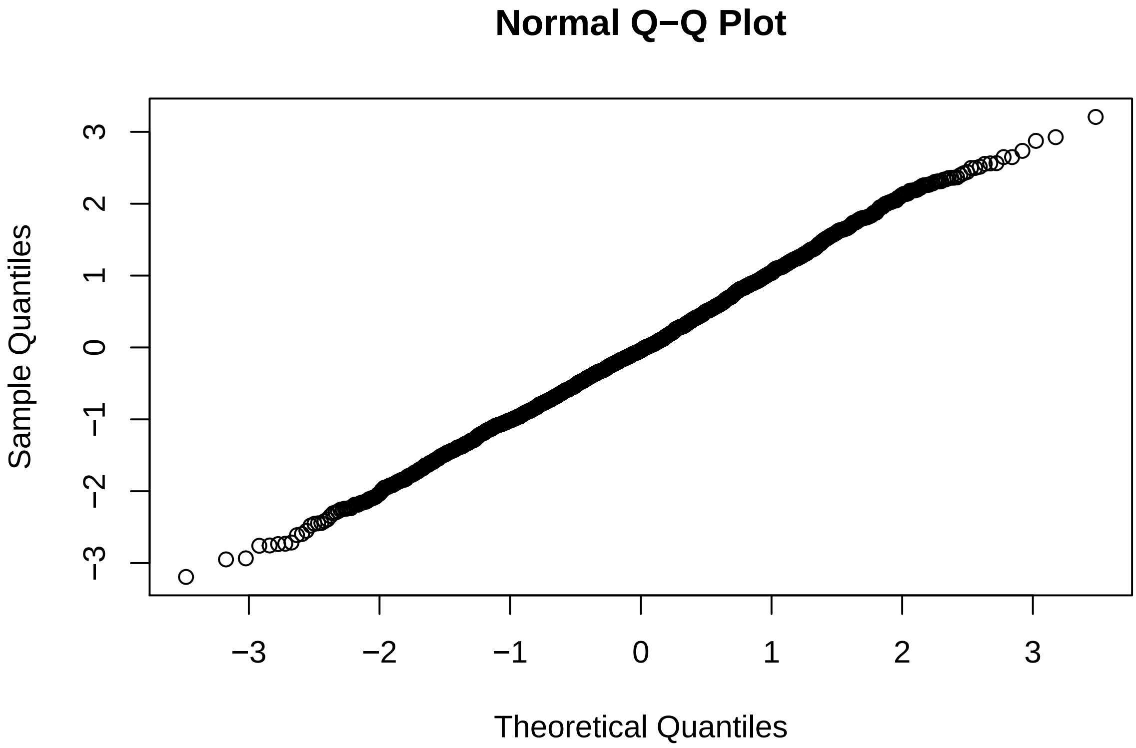
The expression to compute the **answer** is the following $Ans = +0.005562 + 1.032 * f_1 + 0.988 * f_2 + 0.988 * f_4 + 4.94 * f_5$ which is a very good approximation of the one used to produce this random variables.

```
reg_model3<-lm(answer~factor1+factor2+factor3+factor4+factor5, data=dataset)
summary(reg_model3)
```

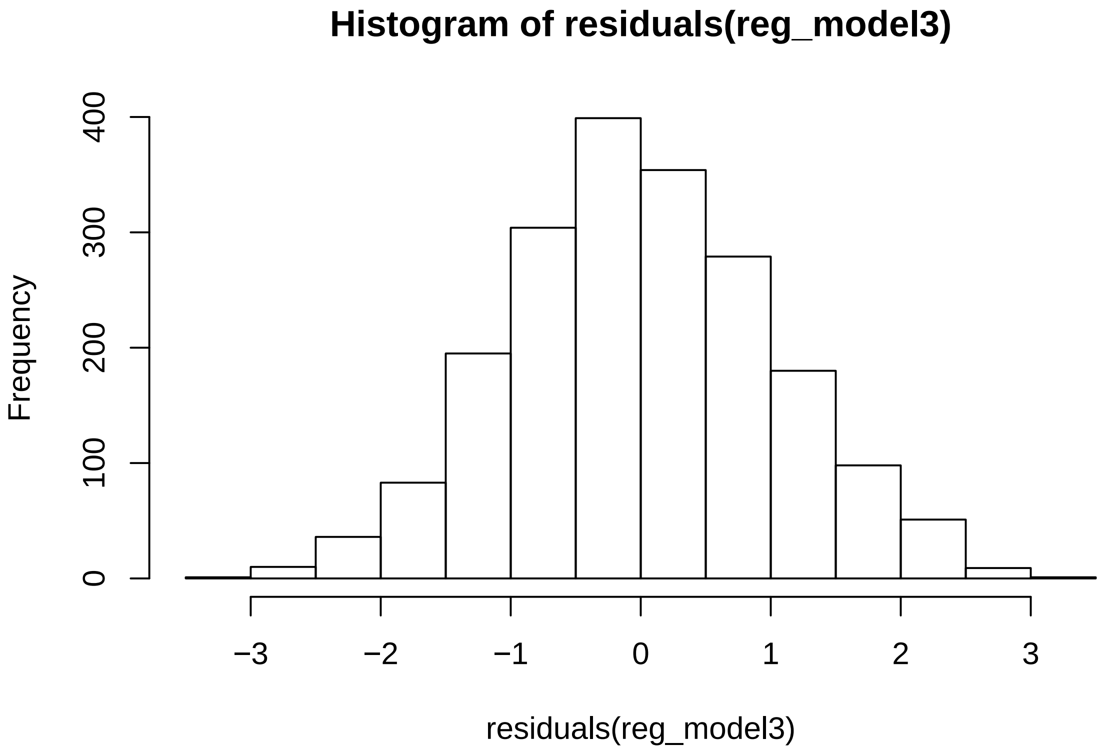
```
##
## Call:
## lm(formula = answer ~ factor1 + factor2 + factor3 + factor4 +
##     factor5, data = dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1928 -0.7032 -0.0417  0.6934  3.2072
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.005562   0.077038   0.072    0.942
## factor1      1.031851   0.038688  26.671   <2e-16 ***
## factor2      0.987810   0.079949  12.355   <2e-16 ***
## factor3     -0.008692   0.011644  -0.746    0.455
## factor4      0.988821   0.037947  26.058   <2e-16 ***
## factor5      4.942126   0.079995  61.780   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.02 on 1994 degrees of freedom
## Multiple R-squared:  0.7327, Adjusted R-squared:  0.732
## F-statistic:  1093 on 5 and 1994 DF,  p-value: < 2.2e-16
```

2.2 Testing Regression Assumptions

```
### 1. Normality of the Error Term
# Using QQ plot
qqnorm(residuals(reg_model3))
```




```
# Using Histogram
hist(residuals(reg_model3))
```

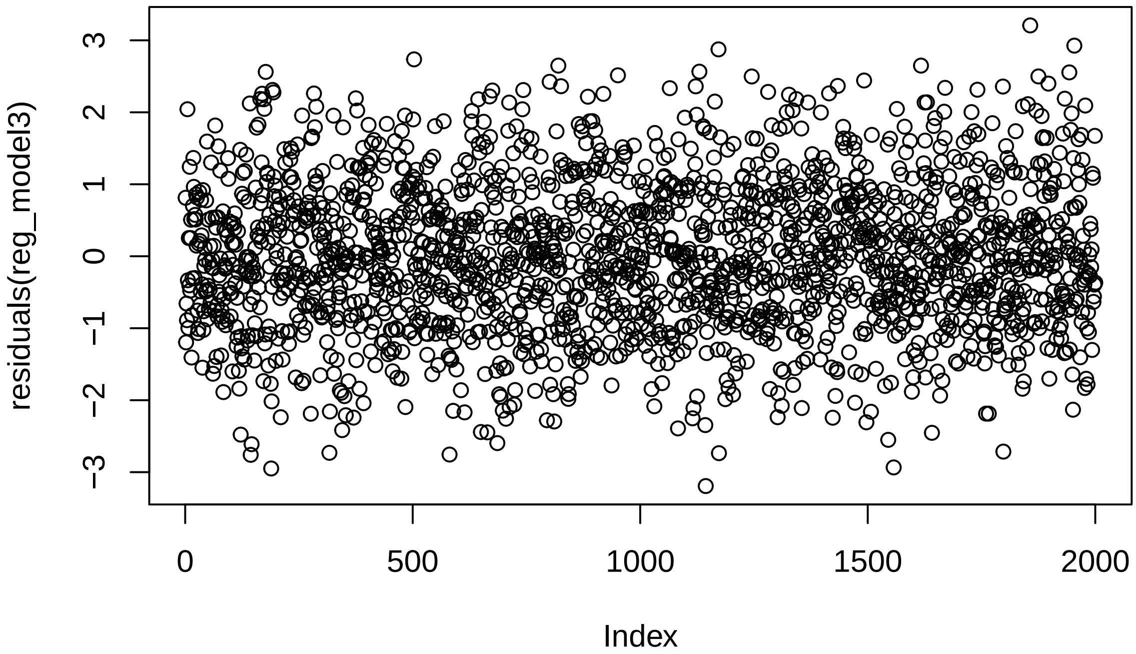



```r
#Shapiro Wilks Test
shapiro.test(residuals(reg_model3))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(reg_model3)
## W = 0.99851, p-value = 0.0734
```

```r
# H_0 is accepted: the error term does follows a Normal distribution (p > 0.05)

### 2. Homogenity of Variance ###
# Residual Analysis #
plot(residuals(reg_model3))
```




```
##Breusch Pagan Test
bptest(reg_model3)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  reg_model3
## BP = 2.5905, df = 5, p-value = 0.7628
```

```
# H_0 is accepted (p>0.05): the homogenity of variances is provided.

### 3. The independence of errors ###
dwtest(reg_model3, alternative = "two.sided")
```

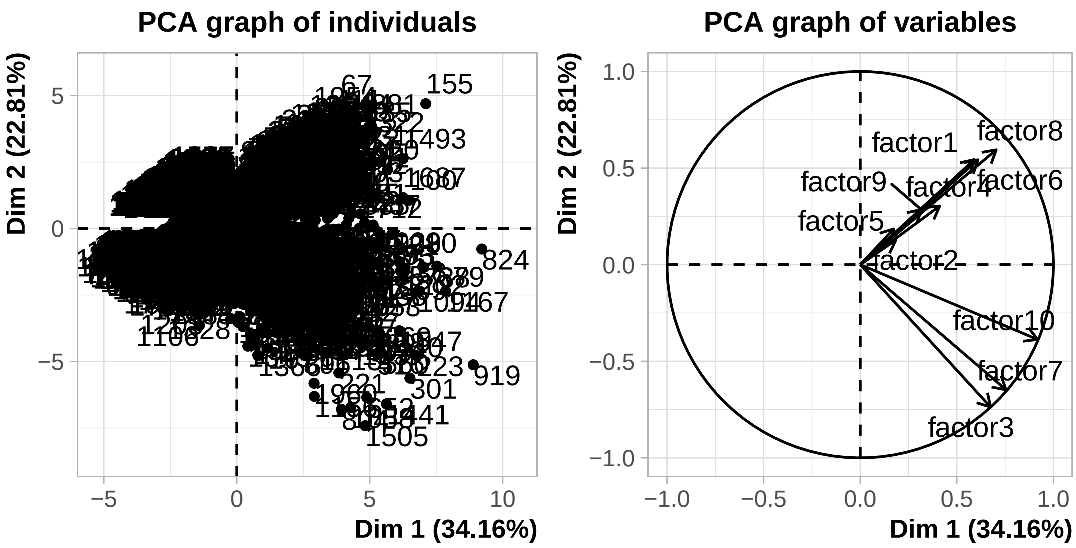
```
##
##  Durbin-Watson test
##
## data:  reg_model3
## DW = 1.9725, p-value = 0.5378
## alternative hypothesis: true autocorrelation is not 0
```

```
# There is not an autocorrelated in the data set (p>0.05).
# The errors/observations are independent.
```

2.3 Principal Component Analysis

Let's use **Principal Component Analysis** technique to analyze the dataset and its factors and extract an expression to predict an answer:

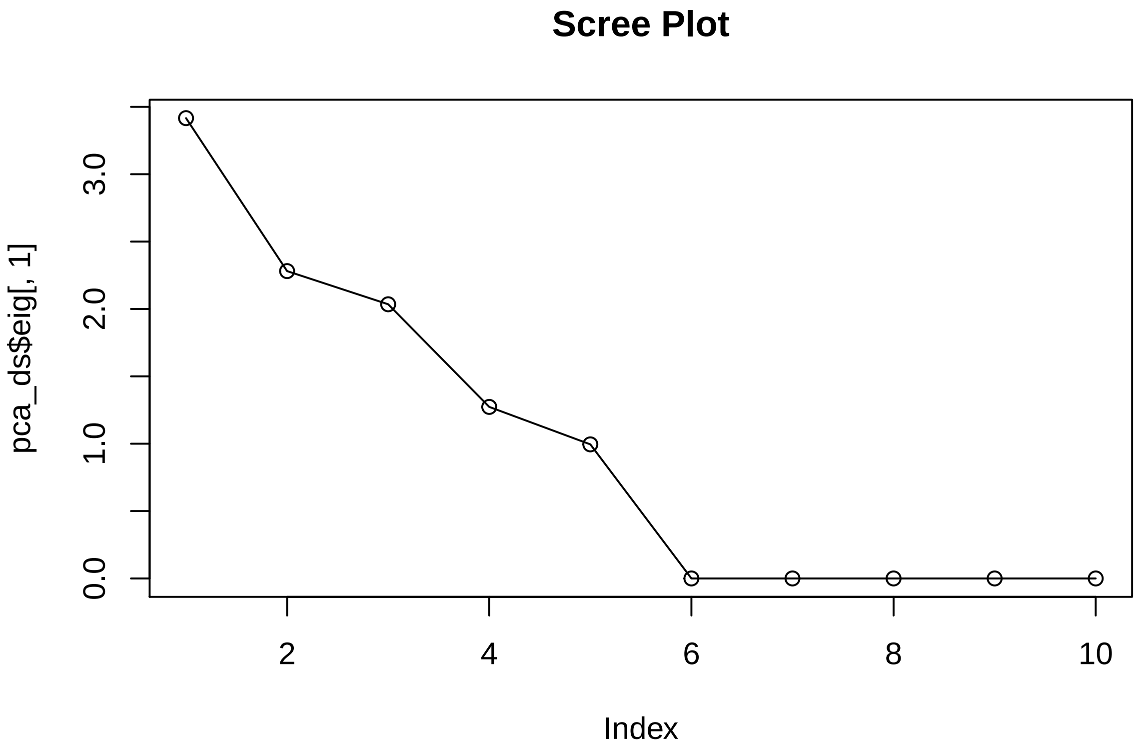
```r
pca_ds<-PCA(dataset[,-answer]) # Remove the dependent variable score.
```




```r
pca_ds$eig # cumulative percentage of variance > 75%
```

```
##           eigenvalue percentage of variance cumulative percentage of variance
## comp 1  3.416127e+00           3.416127e+01                          34.16127
## comp 2  2.280985e+00           2.280985e+01                          56.97113
## comp 3  2.034760e+00           2.034760e+01                          77.31873
## comp 4  1.273078e+00           1.273078e+01                          90.04950
## comp 5  9.950498e-01           9.950498e+00                         100.00000
## comp 6  4.878116e-30           4.878116e-29                         100.00000
## comp 7  2.267126e-31           2.267126e-30                         100.00000
## comp 8  5.748919e-32           5.748919e-31                         100.00000
## comp 9  3.161321e-32           3.161321e-31                         100.00000
## comp 10 1.613421e-32           1.613421e-31                         100.00000
```

```r
plot(pca_ds$eig[,1], type="o", main="Scree Plot")
```



As you may see, the factors that are involved in the answer have the same direction in the plane. On the other hand, the ones that are not related with the answer, have another direction.

In order to extract an expression to predict the answer variable we are going to use a **principal component regression**:

```
### Principal Component Regression
dataset$PC1<-pca_ds$ind$coord[,1]
dataset$PC2<-pca_ds$ind$coord[,2]
dataset$PC3<-pca_ds$ind$coord[,3]
reg_pc<-lm(answer~PC1 + PC2 + PC3, data=dataset)
summary(reg_pc)
```

```
##
## Call:
## lm(formula = answer ~ PC1 + PC2 + PC3, data = dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0129 -0.7970  0.0115  0.7871  3.6616
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.56999    0.02549  179.29   <2e-16 ***
## PC1          0.48107    0.01379   34.88   <2e-16 ***
## PC2          0.54335    0.01688   32.19   <2e-16 ***
## PC3          0.74138    0.01787   41.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.14 on 1996 degrees of freedom
```

```
## Multiple R-squared:  0.6657, Adjusted R-squared:  0.6652
## F-statistic:  1325 on 3 and 1996 DF,  p-value: < 2.2e-16
```

This expression can be used to predict the answer variable.

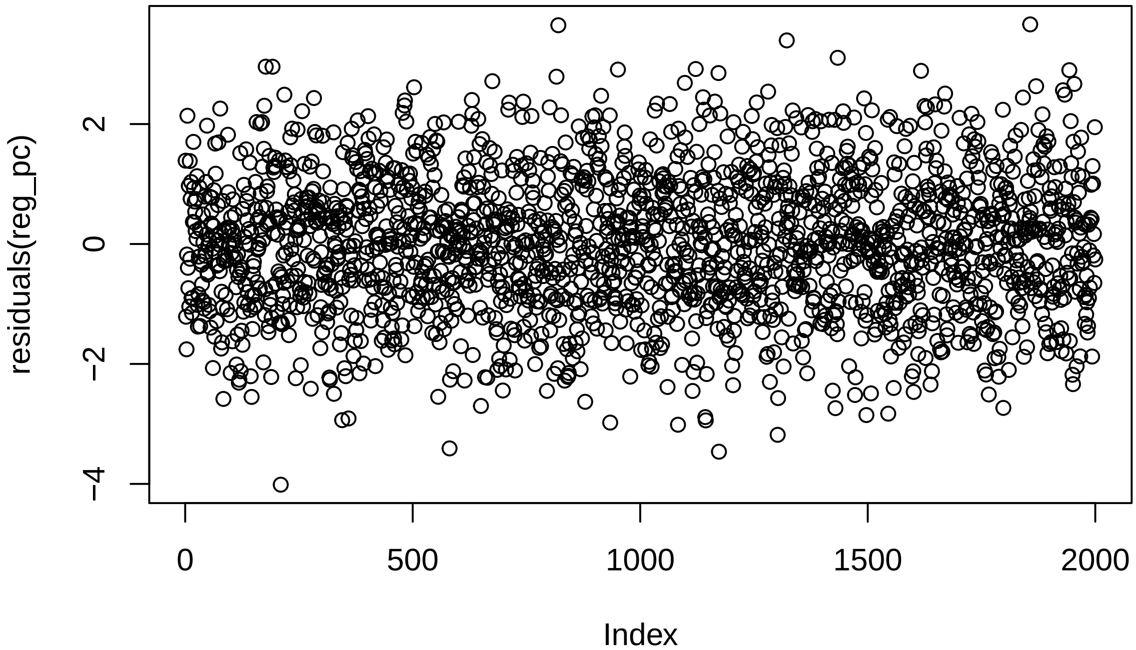
2.4 Testing PCA Assumptions

```r
#1. Normality
#Shapiro Wilks Test
shapiro.test(residuals(reg_pc))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(reg_pc)
## W = 0.99914, p-value = 0.4797
```

```r
# The error term does follow a Normal distribution. (p>0.05)

### 2. Homogenity of Variance ###
# Residual Analysis #
plot(residuals(reg_pc))
```




```r
##Breusch Pagan Test
bptest(reg_pc)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  reg_pc
## BP = 1.6999, df = 3, p-value = 0.637
```

```r
# H0 is accepted (p>0.05).
# The homogenity of variances is provided.
```

```r
### 3. The independence of errors ###
dwtest(reg_pc, alternative = "two.sided")
```

```
##
##  Durbin-Watson test
##
## data:  reg_pc
## DW = 1.9668, p-value = 0.4573
## alternative hypothesis: true autocorrelation is not 0
```

```r
# There is not an autocorrelaiton in the data set (p>0.05).
# The errors/observations are independent.
```

3 Simulate New Data

For this part, we will use the expression obtained from the *LRM* in the previous section and the framework *GPSS* to simulate the data. Here is an snippet 3 of the GPSS code used to generate the model data. The rest of the scripts can be found on the directory `ex3/`.

```
; factor |     min      |     max      |
; -------|--------------|--------------|
; f1     | 0.000836     | 3.483189     |
; f2     | 0.0000648    | 0.9997571    |
; f4     | 0.00125      | 3.43439      |
; f5     | 0.0001625    | 0.9991915    |

; f1 -
; f2 -
; f4 -
; f5 -

; Answer = 0.005562 + 1.032*f_1 + 0.988*f_2 + 0.988*f_4 + 4.94*f_5

GENERATE 1
ADVANCE  (1.032 # 0.000836  + DUNIFORM(RN1, 0, 1)) ;Factor 1
ADVANCE  (0.988 # 0.0000648 + DUNIFORM(RN1, 0, 1)) ;Factor 2
ADVANCE  (0.988 # 0.00125   + DUNIFORM(RN1, 0, 1)) ;Factor 4
ADVANCE  (4.94  # 0.0001625 + DUNIFORM(RN1, 0, 1)) ;Factor 5
TERMINATE 1
```

Listing 1: GPSS script to simulate data

Apart from the previous code, I used included an script to do the replication for the experimental design.

3.1 Validation of the simulation

In order to validate we used some operational validation techniques:

- **Black Box validation**: we compared the real system data with the simulated data to validate its accuracy. We used *Student's t-test* to validate that the model produced from the *LRM* is accurated with respect to the real system. Formaly, *Student's t-test* null hypothesis is that there is no statistical difference between the mean of the given two population.

- **GPSS Traces**: we used the simulation traces to compare the results, and analyze if the logic of the events are coherent with the understanding the experts have of the system.

We used the following *R* script 2 for the *Black Box validation* by the *Student's t-test*. The resulting $p - value = 0.08 \geq \alpha = 0.05$. So we can accept the null hypothesis that is that the populations have the same mean. Hence, our model is a good approximation of the reality.

```
path   <- "/home/arnau/MIRI/SMDE/hw2"
system <- read.csv(paste(path, "ex1/dataset.csv", sep="/"),header=TRUE,sep=",")$answer
model  <- read.csv(paste(path, "ex4/model_data.csv", sep="/"),header=TRUE,sep=",")
t.test(system_small, model)
```

Listing 1: Validation script

4 Design of Experiment

In this section we will use the previously generated model data and analyze the interaction of these records with the answer dependent variable. The code for this section can be found at `ex4/`.

For this part, I wrote another haskell script (see Appendix 4) to run the Yates algorithm on the output of the generated data.

The result of Yates can be found at `ex4/yates.txt`. The file contains 1024 interactions so it is time consuming to analyze by hand. I made a small script to get the most significant factors. As expected, the factors are consistent with the experiment results (see listing 2).

We finish our experimental design by analyzing the results of the factorial experiment. The main factors of our model are:

- Factor 1
- Factor 2
- Factor 4
- Factor 5
- Combinations of the previous ones.

```
[(-),(-),(-),(-),(-),(-),(-),(-),(-),(-)]  =  -5.474452153631347e149
[(-),(-),(-),(-),(-),(-),(-),(-),(-),(+)]  =  -1.515142827886936e149
[(-),(-),(-),(-),(-),(-),(-),(-),(+),(-)]  =  -2.0639841731868795e150
[(-),(-),(-),(-),(-),(-),(-),(+),(-),(-)]  =  3.2096150160054254e150
[(-),(-),(-),(-),(-),(-),(+),(-),(-),(-)]  =  1.7585004371942127e150
[(-),(-),(-),(-),(-),(+),(-),(-),(-),(-)]  =  -3.886988547425833e150
[(-),(-),(-),(-),(+),(-),(-),(-),(-),(-)]  =  1.6898345711885734e150
[(-),(-),(-),(+),(-),(-),(-),(-),(-),(-)]  =  5.717509006989079e149
[(-),(-),(+),(-),(-),(-),(-),(-),(-),(-)]  =  -2.0685619548674406e150
[(-),(+),(-),(-),(-),(-),(-),(-),(-),(-)]  =  -3.0902957714624858e149
[(+),(-),(-),(-),(-),(-),(-),(-),(-),(-)]  =  2.5336790409367517e150
```

Listing 2: Single factors from `ex4/yates.txt`

that are the ones that we used to generate our system answer. **So, we can conclude that our model is an accurate representation of the reality**.

Appendix

Yates implementation in Haskell

```haskell
#!/usr/bin/env nix-shell
#!nix-shell -i runghc
{-# LANGUAGE DerivingStrategies  #-}
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE TypeApplications    #-}
module Main where

import           Control.Applicative          (liftA2)
import           Control.Monad                (replicateM)
import           Control.Monad.Primitive
import qualified Data.ByteString.Lazy         as LBS
import           Data.Csv
import           Data.Foldable                (traverse_)
import qualified Data.List                    as List
import           Data.Monoid
import qualified Data.Vector                  as Vector
import           GHC.Generics
import           Prelude                      hiding (readFile)
import           System.IO
import           System.Random.MWC
import           System.Random.MWC.Distributions

data MinMax = MinMax
  { _min :: Double
  , _max :: Double
  } deriving stock (Show)

data Sign = Minus | Plus
  deriving stock (Enum)

instance Show Sign where
  show Plus  = "(+)"
  show Minus = "(-)"

-- | At most one factor.
amo :: [Sign] -> Bool
amo = (<= 1). getSum . foldMap (Sum . fromEnum)
```

```
-- The Double represents the median of all replications of that combinations o
type Table = [([Sign], Double)]

getTableValues :: Table -> [Double]
getTableValues = fmap snd

-- The Double represents the affect on the answer of the given combination of 
type YatesResult = [([Sign], Double)]

-- (take 10 £ repeat £ MinMax maxBound minBound)
toMinMax :: [[Double]] -> [MinMax]
toMinMax xss =
  foldr go [] (List.transpose xss)
  where
    go xs acc = let x = MinMax (minimum xs) (maximum xs) in x:acc


-- The return are the first column of Yates i.e. the mean of each combination.
factorialDesign :: [MinMax] -> IO Table
factorialDesign minMaxs = do
  gen <- createSystemRandom
  go gen minMaxs [] []
  where
    go :: GenIO -> [MinMax] -> [Sign] -> [Double] -> IO Table
    go gen [] ss vs =
      (\r -> [(ss, r)]) <$> computeAnswer gen vs
    go gen (f:fs) ss vs =
      liftA2 (++)
             (go gen fs (ss ++ [Minus]) (vs ++ [_min f]))
             (go gen fs (ss ++ [Plus])  (vs ++ [_max f]))


    computeAnswer :: GenIO -> [Double] -> IO Double
    computeAnswer gen xs = do
      let formula  = 0.005562 + 1.032*(xs !! 0) + 0.988*(xs !! 1) + 0.988*(xs !!
          -- We add some noise.
          computeFormula =  (formula +) <$> uniformR (0.0, 1.0) gen
      mean <$> replicateM 20 computeFormula
```

```
    mean :: (Foldable t, Fractional a) => t a -> a
    mean xs = sum xs / fromIntegral (length xs)

-- Yates: returns the effect on the answer of each factor.
yates :: Table -> YatesResult
yates xs =
  zip (fst <$> xs)
      (go (length xs + 1) (snd <$> xs))
  where
    go :: Int -> [Double] -> [Double]
    go 0 ys = let l = fromIntegral $ length xs
               in (head ys / l) : ((/ (l/2)) <$> tail ys)
    go n ys = go (n - 1) (add2 ys ++ subtract2 ys)

    add2 :: Num a => [a] -> [a]
    add2 []             = []
    add2 (x1 : x2 : xs) = (x1 + x2) : add2 xs

    subtract2 :: Num a => [a] -> [a]
    subtract2 []             = []
    subtract2 (x1 : x2 : xs) = (x2 - x1) : subtract2 xs

saveModelData :: FilePath -> Table -> IO ()
saveModelData fp table =
  let modelData = Only <$> getTableValues table
   in LBS.writeFile fp (encode modelData)

writeResults :: FilePath -> [([Sign], Double)] -> IO ()
writeResults fp result =
  withFile fp WriteMode (\h -> traverse_ (writeRow h) result)
  where
    writeRow h (ss, result) = hPutStrLn h $ show ss ++ "  =  " ++ show result

main :: IO ()
main = do
  putStrLn "=== Factorial design started ==="
  bs      <- LBS.readFile "../ex1/dataset.csv"
  putStrLn "- Read dataset"
  dataset <- either fail (pure . fmap init . Vector.toList) $ decode @[Double] H
  putStrLn "- Decode dataset"
  table <- factorialDesign $ toMinMax dataset
```

```
putStrLn "- Data and Experiment prepared"
saveModelData "model_data.csv" table
putStrLn "- Data saved"
let result = yates table
writeResults "yates.txt" result
putStrLn "- Yates result saved"
writeResults "yates_single_factor.txt" (filter (amo . fst) result)
putStrLn "- Yates single factor saved"
putStrLn "=== Successfuly completed ==="
```