Universitat Politècnica de Catalunya

SMDE Second Assignament

Design of Experiments

Alejandro Montero Rivero
30-11-2016

Contenido

1- Random Number Generator (RNG)

In the first section of this practice session we are going to create a random number generator. Randomness is a common occurrence in real world and also a really useful concept in statistics. For that reason, we are going to simulate random numbers in specific situations.

Random Number Generator are random variables generated following a uniform distribution U[0,1]. As the distribution suggest all numbers do not follow a specific pattern which indicates all possible combinations have the same probability, this is exactly what we want to avoid generating predictable values.

In this section after creating the Random Number Generator with R code, afterwards we are going to analyse whether or not it gives enough randomness. A set of samples created with this RNG will be used to do the analysis.

1.1- Park-Miller Random Number Generator

As suggested we are picking an already created definition of an RNG, the reason its pretty straightforward, defining and testing a new RNG is a titanic task way out of the scope of the course. A congruential generator called Park-Miller is selected, it is defined by the recurrence relation:

$$X_{n+1} = (a * X_n + C) mod\ m$$

As the relation indicates, all numbers are generating using the previous value. More specifically, the components that are part of the generator are the followings:

- X_{n+1}: Next number to calculate.
- X_n: Previous number calculated.
- $mod\ m$: Modulus of the congruence.
- a: Multiplier.
- C: Number to increment.
- X_0: The seed, or, the first value of the sequence.

The R code to define this congruential generator is the following:

```
#####################################
# Random Number Generator
#####################################
parkMiller <- function(k,mul,increm,seed){
  X2=0
  X1=seed
  out = 0
  sortOut = c()
  for (i in 1:k) {
    X2 = (mul*X1 + increm) %% (2147483647)
    out = X2 / (2147483647)
    sortOut <- c(sortOut,out)
    X1 = X2
  }
  return(sortOut)
}
```

1.2- Testing the Random Number Generator

At this very same moment we already have the Random Number Generator implemented and working, but we still cannot consider it for production as we do not know if the randomness of the created samples is correct. With the following tests, we want to check whether or not this observation follow a uniform distribution and no correlation exists between the numbers of the sequence.

To prove the correctness of the RNG one stream of 5000 observations is created and the Chi-square test is performed to probe if the stream follows a uniform distribution. Not only we want that the whole stream follows a uniform distribution but also all sub-streams must follow it. To check that, we will create a set of sub-streams from the initial stream, reorder them and perform the Chi-square test once again.

As in the last assignment to perform the Chi-square test we have to divide the stream into diverse intervals, luckily, we already know that these intervals will have the same density as our stream follows a normal distribution (That is our H_0).

The resulting histogram after distributing the data among the intervals is the following:

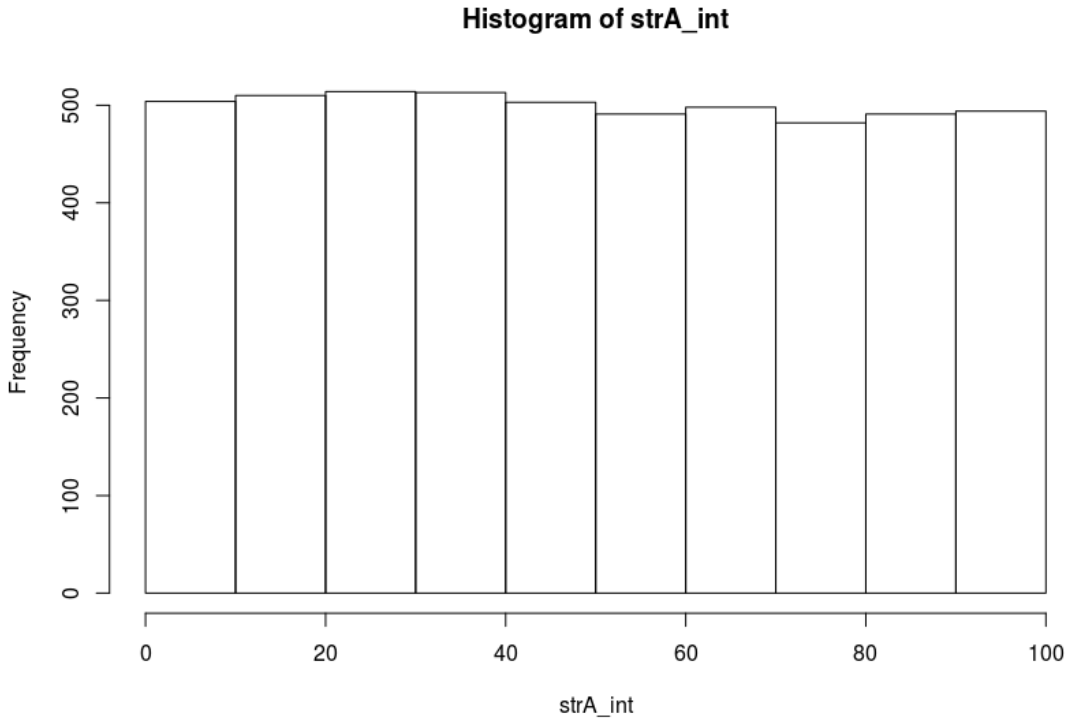

Figure 1. Stream A distributed among intervals

As we can clearly see just by looking at the histogram the distribution of the observations is almost completely uniform. Though the clear uniform behaviour of the stream A we still cannot conclude our H_0 is true, so we can use the X^2 to prove it. To do so we are going to need the formula that defines the X^2distribution:

$$X^2 = {}^{k}/_{n} \sum_{j=1}^{k} (f_j - {}^{n}/_{k})^2$$

We already know the theoretical value of $\frac{k}{n}$ as k is our number of intervals and n our number of pseudo-random observations. So, our solution is to create a new vector containing our theoretical value and perform the X^2 against our observations. The next snippet exemplifies the procedure:

```
intervals <- seq(from = 0, to = 1, by = 0.01)
strA_int <- cut(x = strA, breaks = intervals, right = TRUE, labels
= FALSE)


theor_val = length(strA)/(length(intervals) - 1)
theor_val_int = rep(theor_val,length(strA_int))
tbl <- table(strA_int, theor_val_int)
chisq.test(tbl)
```

The final result for the test:

```
      Chi-squared test for given probabilities

data:  tbl
X-squared = 81.36, df = 99, p-value = 0.9013
```

As our p-value is way higher than 0.05, our confidence level, we can conclude out observations are uniformly distributed and in extension that our Random Number Generator does, for now, a really good job. Though this result is excellent, with just a single test we cannot still conclude our RNG really creates actual random numbers, so our second test consists on checking if the inner sub-streams are also uniformly distributed.

1.3- Inner streams test

This second tests consists in breaking the Stream A in several sub-streams and perform the X^2 as we have done in the previous section. More specifically the stream is going to be divided in 3 sub-streams, not only that but to do the analysis fairer we will also sort the observations. The snipper below shows how to do this procedure with one sub-stream:

```
strA_B = strA[1:1500]
srtA_B_tmp = strA_B[1000:1500]
srtA_B_tmp2 = strA_B[1:1000]
strA_B = c(srtA_B_tmp, srtA_B_tmp2)
```

Once we have a sorted sub-stream we can do the X^2 test exactly as in the previous section. The histograms for each sub-stream are the following:

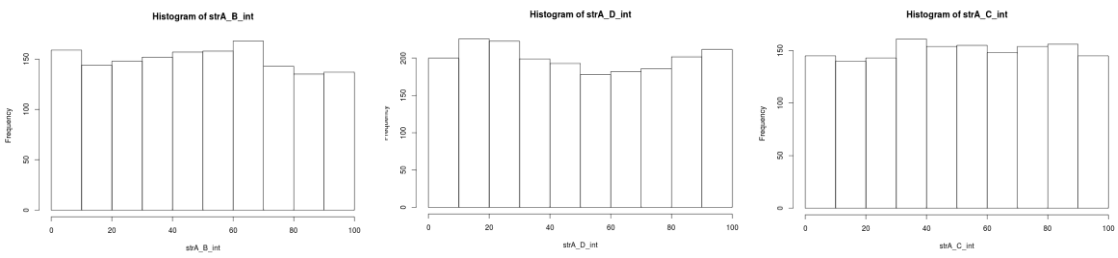

Figure 2. Sub-Stream distributed among intervals

And as before the results of the X^2 test for each stream:

```
        Chi-squared test
for given probabilities

data:  tblAB
X-squared = 110.53, df =
99, p-value = 0.2015
```

```
        Chi-squared test
for given probabilities

data:  tblAC
X-squared = 103.06, df =
99, p-value = 0.3699
```

```
        Chi-squared test
for given probabilities

data:  tblAD
X-squared = 110.59, df =
99, p-value = 0.2002
```

As we can see the tests results in complete success for the all three sub-streams. For this assignament this two set of tests are more than enough to demonstrate the validity of this Random Number Generator, in a real world escenario though, this tests won't be nearly enough to accept our hypothesis that the RNG uniformely distributes all created observations, thus a whole new battery of tests are needed.

1.4- Conclusions

Random Number Generators have applications in lots of different areas such as computer simulation, cryptography, randomized design and statistical sampling. In this section, we have implemented with R the definition of the Park-Miller congruential RNG.

One of the main concerns when using one RNG is if it can provide actual random observations, and of course, before using one solution on production it is mandatory to test it. For that reason, to check if the RNG was capable of providing enough randomness two tests are performed. Those tests consist of checking if a single great stream of observations and its inner sub-streams can be uniformly distributed.

The answers the X^2 tests provide, demonstrates our implementations are good enough and the observations generated by the RNG can be considered random.

2- Data simulation

The main objective of this assignment is to perform experiments with data, and in order to do so, we first need that data. Observations can be gathered from the real world, but in our case, we are going to create said data artificially.

Initially we are going to define five factors which consists in simulated data of about 100 observations of different distributions. Six other factors are then defined in function of the previous ones. Next table describes each factor:

Factor	Definition	Factor	Definition
F1	$\sim N(0,1)$	F6	$F1 + 5 * F3$
F2	$\sim N(10,5)$	F7	$F5 + 2 * F3$
F3	$\sim Exp(3)$	F8	$F5 + F2$
F4	$\sim Exp(10)$	F9	$F4 + 3 * F2$
F5	$\sim U(10,20)$	F10	$F1 + F5$

In order to analyse if our future experiments provide good solutions we need to define an answer variable, which is composed by a subset of the previously defined factors. The following expression defines the answer variable:

$$AnsVar = F1 + 2F3 - 5F6 - 4F4 + F8 - 2F10 + F_{N(5,2)}$$

Next snippet exemplifies how to create the answer variable:

```
dat <-matrix(nrow=100, ncol=10, dimnames=names)
dat[,1]<-rnorm(100,mean=0,sd=1)
dat[,2]<-rnorm(100, mean=10,sd=5)
dat[,3]<-rexp(100, rate=3)
dat[,4]<-rexp(100, rate=10)
dat[,5]<-runif(100, min=10, max=20)
dat[,6]<-dat[,1] + 5*dat[,3]
dat[,7]<-dat[,5] + 2*dat[,3]
dat[,8]<-dat[,5] + dat[,2]
dat[,9]<-dat[,4] + 3*dat[,2]
dat[,10]<-dat[,1] + dat[,5]
answer_var = dat[,1] + 2*dat[,3] - 5*dat[,6] - 4*dat[,4] + dat[,8] - 2*dat[,10] +
rnorm(100,5,2)
dat_ans = cbind(dat,answer_var)
```

3- Linear model

Next step in the process is create a model capable of generating new data. As we are assuming we do not know anything about the dataset, we could use some tools learned in the previous assignment to discover the best way to define our model.

3.1- PCA

PCA can be a really useful tool to analyse the relation between the factors and the answer variable.

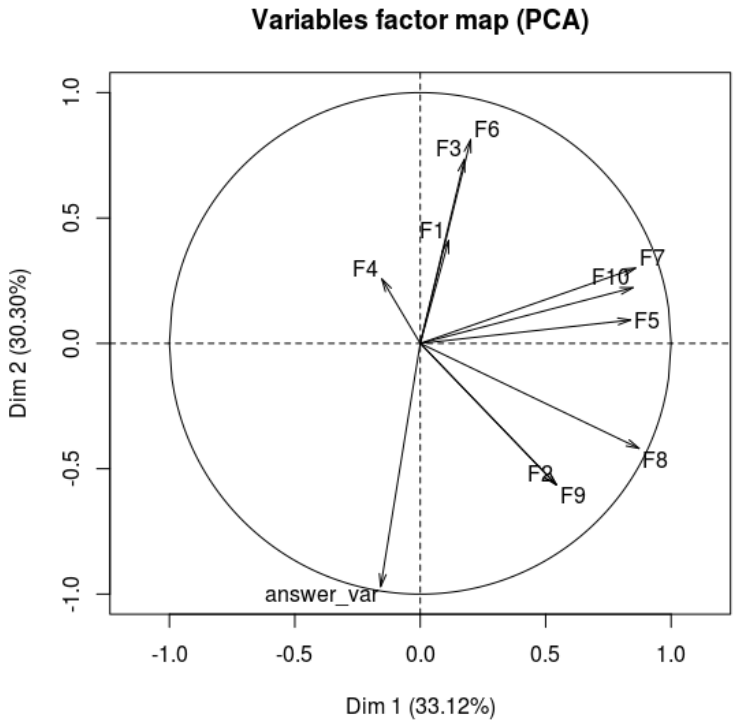

Figure 3 Factor's PCA

From this PCA we can extract very useful information. First it seems that exists one great subset that groups most of the factors. In this case, we can already see that factors 1, 3, 5, 6, 7 and 10 have strong correlation between each other. Three other factors: 2, 8 and 9, are grouped in a second subset. Factor 4 lies out of the two subsets though it shows a heavy correlation with factors 8, 9 and. Finally, for the answer variable we can see it has a strong correlation with factors 1, 3 and 6.

Taking into account we already know how all factors are described we can see the PCA makes a lot of sense as, for example, F6 is formed by a combination of factor 1 and 3, while factor 10 and 7 is created using factor 5. That very same behaviour can be seen on the second subset for factor 8 and 9 which are a combination of factor 2.

3.2- Regression Model

Using the informatio, the PCA analysis have provided us, we could create a model using just the factors 1, 3 and 6 though, maybe, we could still be missing some information. For that reason, we have created some combinations of models until getting the one with best R-squared statistic, using as well, the information provided by PCA.

Initially the model containing all factors shows the following summary:

```
lm(formula = answer_var ~ F1 + F2 + F3 + F4 + F5 + F6 + F7 +
    F8 + F9 + F10, data = dat_frame)

Residuals:
    Min      1Q  Median      3Q     Max
-3.0443 -1.2030 -0.0565  1.2270  3.6584

Coefficients: (5 not defined because of singularities)
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.68632    1.07065   3.443  0.00086 ***
F1           -5.93981    0.18699 -31.766  < 2e-16 ***
F2            1.01923    0.03834  26.585  < 2e-16 ***
F3          -22.49816    0.42345 -53.130  < 2e-16 ***
F4           -2.99067    2.43678  -1.227  0.22277
F5           -0.95684    0.06209 -15.411  < 2e-16 ***
F6                 NA         NA      NA       NA
F7                 NA         NA      NA       NA
F8                 NA         NA      NA       NA
F9                 NA         NA      NA       NA
F10                NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.726 on 94 degrees of freedom
Multiple R-squared:  0.9822,     Adjusted R-squared:  0.9812
F-statistic:  1035 on 5 and 94 DF,  p-value: < 2.2e-16
```

Using all factors gives, obviously, a very huge R-squared value, this means the data fits the regression line, a value close to 100% indicates that the model explains all the variability of the response data around its mean. Though we want a model that fits the data as best as possible, this model contains factors that do not explain any information.

First of all, we are going to eliminate from the model all factors that shows an NA. Factors that receive an NA indicates they are linearly related to the other variables. We expected that behaviour as factors 6, 7, 8, 9 and 10 are indeed created using all other factor, aka, are linearly related to factors 1, 2, 3, 4, 5

Second step is to eliminate those variables that add little to the model, in our case the only one to eliminate is factor 4. PCA already gave us that insight, as we are modelling the answer variable we could previously see it was correlated with factors 1,2,3 and 5 while factor 4 was not even close to create a small correlation.

Finally, we decided to create the regression model using factors 1,2,3 and 5. The summary of the model gives outputs the following:

```
lm(formula = answer_var ~ F1 + F2 + F3 + F5, data = dat_frame)

Residuals:
    Min      1Q  Median      3Q     Max
-3.2269 -1.3319 -0.1148  1.3163  3.8218

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.30391    1.02703   3.217  0.00177 **
F1          -5.91993    0.18678 -31.695  < 2e-16 ***
F2           1.02650    0.03798  27.028  < 2e-16 ***
F3         -22.59623    0.41695 -54.194  < 2e-16 ***
F5          -0.95156    0.06211 -15.322  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.731 on 95 degrees of freedom
Multiple R-squared:  0.9819,     Adjusted R-squared:  0.9811
F-statistic:  1286 on 4 and 95 DF,  p-value: < 2.2e-16
```

This combination of factors for creating the model seem to fit really well the data, as it can explain the 98% of variability. The model not only contains useful variables excluding those that are linear combination of others but also all factors that add noise have been excluded.

It's important to note that if no linear relationship between the variables, we should expect the slope to be zero. Initially, we could neglect this possibility as we have created the data and we already know there is a clear relationship between the variables. As in a real-world scenario we would not have as much insight about the data we can check the p-value of the t-statistic. Finally, as this value is really close to 0, we can assume a linear relation actually exists.

One last mandatory step is to ensure our model does not fail any of our assumptions:

Independent Observations

To test if the observations within each sample are independent we can use the Durbin-Watson test. As we can see we pass this condition.

```
	Durbin-Watson test

data:  RegModel.2
DW = 2.1871, p-value = 0.8305
```

Normality Test

We also need to test if the populations follow a normal distribution. The saphiro test shows that we pass this second condition

```
	Shapiro-Wilk normality test

data:  residuals(RegModel.2)
W = 0.98092, p-value = 0.1568
```

Equal variances

Finally, we have to test if the observations of the samples have equal variances. Breusch pagan test shows that we pass the final test.

```
	studentized Breusch-Pagan test

data:  RegModel.2
BP = 1.8987, df = 4, p-value = 0.7544
```

3.3- Analysis of the model

After creating the model, we want to check how it performs and how good the predictions are. To test that we have created a secondary data set. Using a prediction interval of 95% and the following code we could create the point cloud chart shown below.

```
prediction <- predict(object=RegModel.2,newdata = dat_frame,interval="prediction")


plotCI(prediction[,1],li=prediction[,2],ui=prediction[,3],pt.bg="green",ylab="pred
iction"
       ,xlab="observations",pch=21, main="Predicted values")
points(dat_frame$answer_var, col="red")
```

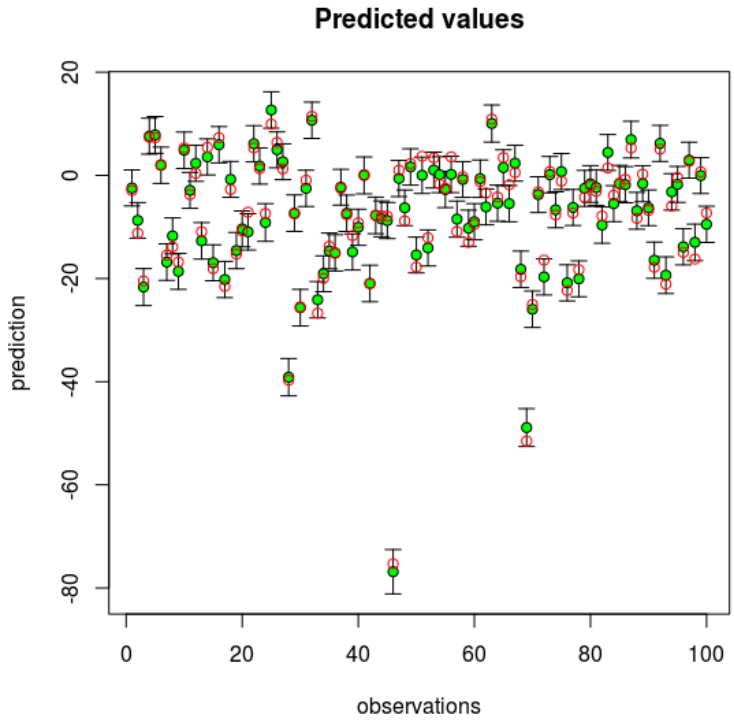

Figure 4. Predicted values vs observed values

As we can see from the point cloud the prediction works flawlessly for our dataset. Of course, we expected that behaviour as we are testing a model against the dataset used to create it and, usually, the prediction fits the observations. As we don't have more date to test the model against, we will accept this as good enough and thus, that our model is excellent for creating new data.

4 – Design of Experiments

After creating some artificial data and designing a model capable of obtaining new data, now we are in position of creating an analysis of the factors and conclude which of them are contributing the most. To do so we are going to apply the Yates algorithm, which involves following a series of steps.

4.1- Objective of the experiment

The main drawback we are facing at the moment is that our data has no real semantics, or rather, what is the actual changes and implications of each factor. Just to exemplify, we may want to maximize the productivity in our office and some factors could be adding a secondary screen for the workers, offer scholarships, improve the air-conditionate system and so on. In our case, though, we don't have the luxury to know if we want to maximize or minimize and for that reason we are going to consider both options.

Our final objective is to check which factors contribute the most to the maximization and minimization of the answer variable.

4.2- Variable selection

In a real-world scenario, we would have, probably, hundreds of factors conditioning our objective and as a result thousands of experiments would be in order. So, it is mandatory to reduce the complexity of the problem. In this assignment, we are emulating a real-world scenario, and for that reason we are going to reduce the problem complexity as much as we can.

In the previous section, we have created a model using just a subset of the factors. The main two reasons why we have eliminated six out of the ten original factors are the following:

- Some factors are a linear combination of other factors; thus, they do not add any useful insight in the variability of the data.
- Factors that add very little information to the variability of the data. The way the observations of those factors are created and how they are used to define the answer variable are the reasons they may not provide any usefulness.

As a reminder, we finally discovered that just factors 1,2,3 and 5 are useful for modelling the answer variable, and for that very same reason we are going to automatically eliminate the remaining factors from our next experiments, as we already know they provide close to non-explanation of the variability.

Initially we would need to perform a total of 2^k experiments, being k the number of factors in our dataset. That results in a total of $2^{10} = 1024$, after the reduction of the number of factors though, we only need to perform $2^4 = 16$ experiments, which implies a reduction of nearly the 98% of experiments. We can consider this a great success.

4.3 - Experimental design

In order to apply Yates, we need to create a factorial design. To fix the problem only two levels are going to be selected per each factor, those levels could be the max minimum observations though most probably we would be picking outliers that are not correctly representing the population behaviour. For that reason, we are picking the first quartile as our minimum and the third quartile as the maximum. With the help of summaries and boxplots we can see the distribution of data in each factor.

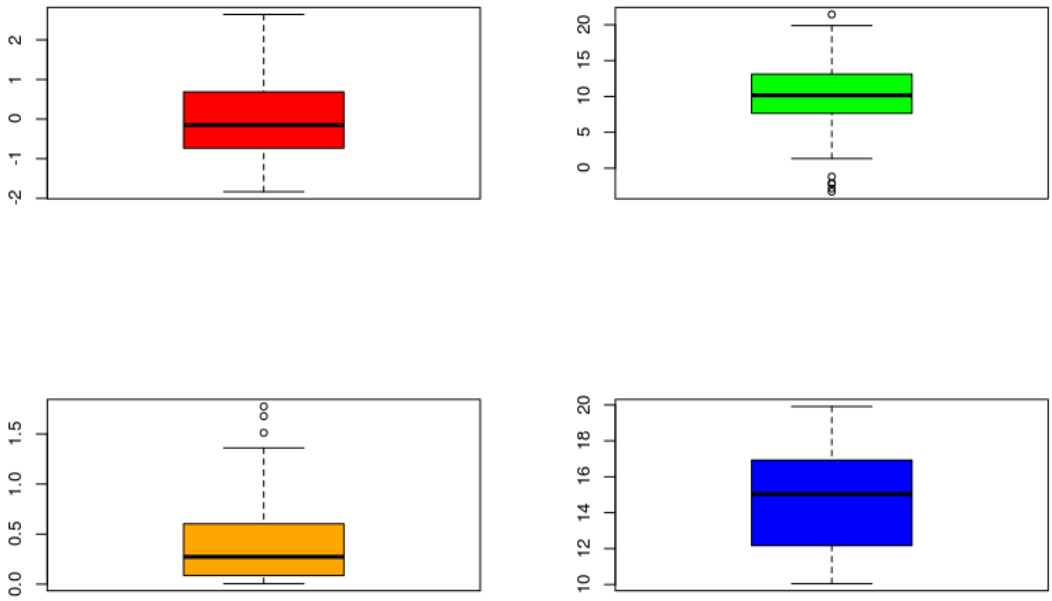

Figure 5. BoxPlots of the four Factors

The final minimum and maximum values for the four factors are the following:

Factor	1st Quartile	3rd Quartile
F1	-0.724900	0.673500
F2	7.655	13.110
F3	0.086720	0.602800
F5	12.22	16.88

Finally, the experiments that are going to be performed are the followings:

Experiment	Factor 1	Factor 2	Factor 3	Factor 5
1	-	-	-	-
2	-	-	-	+
3	-	-	+	-
4	-	-	+	+
5	-	+	-	-
6	-	+	-	+
7	-	+	+	-
8	-	+	+	+
9	+	-	-	-
10	+	-	-	+
11	+	-	+	-
12	+	-	+	+
13	+	+	-	-
14	+	+	-	+
15	+	+	+	-
16	+	+	+	+

Once defined all experiments to perform we need to calculate a predicted answer variable value for each iteration. The prediction as usual results in an interval that specifies three important aspects:

- **Fitted value.**
- **Upper boundary.**
- **Lower boundary.**

To perform the prediction a new data set has been created with the quartiles of each factor, we will use the 1st or 3rd quartile depending on the experiment number we are predicting. The snippet below exemplifies how to do this action in R.

```
> dimnames = list(seq(1:16),
                  c("F1","F2","F3","F5"))
> dat_predict <-matrix(nrow=16, ncol=4, dimnames=dimnames)
>dat_predict[,1]<- c(-0.72,-0.72,-0.72,-0.72,-0.72,-0.72,-0.72,-
0.72,0.67,0.67,0.67,0.67,0.67,0.67,0.67,0.67)
> dat_predict[,2]<-
(7.66,7.66,7.66,7.66,13.11,13.11,13.11,13.11,7.66,7.66,7.66,7.66,13
.11,13.11,13.11,13.11)
> dat_predict[,3]<-
c(0.09,0.09,0.6,0.6,0.09,0.09,0.6,0.6,0.09,0.09,0.6,0.6,0.09,0.09,0
.6,0.6)
> dat_predict[,4]<-
c(12.2,16.9,12.2,16.9,12.2,16.9,12.2,16.9,12.2,16.9,12.2,16.9,12.2,
16.9,12.2,16.9)
> prediction <- predict(object=RegModel.2, newdata =
data.frame(dat_predict),interval="prediction")
```

This prediction will follow a Gaussian distribution with the $\mu = fitted\ value$. As we are using a prediction model we already know that the upper boundary is defined using the following formula:

$$UpperBoundary = \mu + 1.64\,\sigma$$

And therefore, we can deduce the formula to calculate the standard deviation:

$$\sigma = \frac{UpperBoundary - \mu}{1.64}$$

To add even more signification two actions are also performed. First, our answer variable will be the union of the prediction mean and its standard deviation alongside a random probability ranging from 0 to 1. Second, three replications have been made to calculate the answer variable, the mean of the three replications will be the final answer variable that we are going to use in the Yates algorithm.

Finally, with all the means for our answer variable, we can finally run the Yates algorithm using a spreadsheet. The final results are shown below.

F1	F2	F3	F5	VALUES				Fit	lower	upper	μ	StDev	Rand prob.	x1	x2	x3	Mean	StDev	Yates-1	Yates-2	Yates-3	Effects	
-	-	-	-	-0.72	7.66	0.087	12.22	2.3559891	-16769415	6.38891957	2.3559891	2.459103945	0.777139138	3.504034005	0.662676222	1.950789971	2.039166733	0.976578182	2.271019544	-20.1090152	-22.8029565	-1.42518478	Mean
-	-	-	+	-0.72	7.66	0.087	16.88	-2.2207526	-6.22221784	1.78067325	-2.2207526	2.439893811	0.970574164	2.38893822	-0.45859802	-1.23478177	0.231852811	1.438056939	-22.3800347	-2.6939413	-93.9042465	-11.7380308	F5
-	-	+	-	-0.72	7.66	0.602	12.22	-9.4853938	-13.5096092	-5.46117846	-9.4853938	2.453789841	0.607816162	-8.81395913	-8.32334239	-8.92282135	-8.68670762	0.24224349	11.04031146	-58.1715231	-19.7632182	-2.47040228	F3
-	-	+	+	-0.72	7.66	0.602	16.88	-14.0621355	-18.0734842	-10.0504293	-14.0621355	2.446162293	0.881659078	-11.1675396	-14.0585076	-15.853934	-13.6933271	1.683858305	-13.7342528	-35.7327233	-13.0109805	-1.62637256	F3*F5
-	+	-	-	-0.72	13.11	0.087	12.22	7.8381458	3.7999952	11.87633959	7.8381458	2.462313287	0.719925241	9.272737382	7.920901745	10.43610565	9.209914925	0.85934212	-19.1806048	-6.81393337	-49.4256185	-6.17820231	F2
-	+	-	+	-0.72	13.11	0.087	16.88	3.2614042	-0.7434482	7.26625657	3.2614042	2.441983152	0.013492984	-2.13958051	4.155204176	3.475565947	1.830396536	2.646651367	-38.9909184	-12.9492848	-42.6732148	-5.33415184	F2*F5
-	+	+	-	-0.72	13.11	0.602	12.22	-4.0032371	-8.0406453	0.03417108	-4.0032371	2.461834256	0.438836713	-4.38216072	-3.06360459	-4.80096416	-4.08224316	0.679092377	-6.4349111	-8.59956398	-1.38955359	-0.1736942	F2*F3
-	+	+	+	-0.72	13.11	0.602	16.88	-8.5799787	-12.603049	-4.55690839	-8.5799787	2.453091652	0.033126866	-13.0855798	-5.11847225	-10.7519767	-9.6520096	3.022358239	-29.2978122	-4.41141653	-4.19224787	-0.52403098	F2*F3*F5
+	-	-	-	0.67	7.66	0.087	12.22	-5.9188001	-9.9528403	-1.88476002	-5.9188001	2.459780537	0.548198672	-5.62089219	-5.9714588	-13.0832319	-8.22519431	3.238691755	-1.80731392	-24.6510542	17.41507385	2.176884232	F1
+	-	-	+	0.67	7.66	0.087	16.88	-10.4955418	-14.4975118	-6.4935717	-10.4955418	2.440225671	0.104613911	-13.5597126	-6.66231352	-8.58115024	-9.60105877	2.639102526	-5.00661945	-24.7745642	22.43879977	2.804849971	F1*F5
+	-	+	-	0.67	7.66	0.602	12.22	-17.760183	-21.7784503	-13.7419157	-17.760183	2.450162963	0.971404308	-13.1003521	-16.584102	-17.9663742	-15.8836094	1.855504877	-7.37951839	-19.8103136	-6.13535146	-0.76691893	F1*F3
+	-	+	+	0.67	7.66	0.602	16.88	-22.3369246	-26.3420904	-18.3317588	-22.3369246	2.44217425	0.503978988	-22.3125663	-20.7379178	-26.2714427	-23.1073089	2.109422509	-5.56976645	-22.8629011	4.188147445	0.523518431	F1*F3*F5
+	+	-	-	0.67	13.11	0.087	12.22	-0.4366434	-4.4745214	3.60123456	-0.4366434	2.462120707	0.327998023	-1.53339009	-6.09168016	3.617053656	-1.33600553	3.302039459	-1.37586446	-3.19930553	-0.12350998	-0.01543875	F1*F2
+	+	-	+	0.67	13.11	0.087	16.88	-5.013385	-9.01733451	-1.0094249	-5.013385	2.441439085	0.518272693	-4.90152101	-4.27737293	-3.92962825	-4.36950739	0.354675743	-7.22369952	1.809751943	-3.05258754	-0.38157344	F1*F2*F5
+	+	+	-	0.67	13.11	0.602	12.22	-12.2780263	-16.3080791	-8.2479735	-12.2780263	2.457349268	0.450686885	-12.5825561	-17.4421103	-11.85518	-13.9599488	2.321441012	-3.03350186	-5.84783506	5.009057474	0.626132184	F1*F2*F3
+	+	+	+	0.67	13.11	0.602	16.88	-16.8547679	-20.8698844	-12.8396514	-16.8547679	2.448241744	0.45766005	-17.1150908	-11.7901866	-17.108313	-15.3378635	2.365117897	-1.37791467	1.655587189	7.503422246	0.937927781	F1*F2*F3*F5

4.5 - Final analysis

If we take a look at the results provided by Yates algorithm we can see the mean, the basic experiment that adds no factors, has a value of ~ -1.4, so, any factor used that results in a value bigger that the mean means it's a great choice for maximizing our answer variable, aka, if the semantics of our problem where to increase performance of an office, any given factor above the mean would theoretically increase the performance.

The very same methodology can be applied for the minimization. For instance, if we want to reduce the time to produce a screw in a factory, any given factor lower than the mean would reduce the time to produce it.

Starting with the maximization we can clearly see the main factor contributing to it is the Factor 1. All other combinations with other factors results in lower improvement, so, in this scenario if we want to maximize the answer variable we would only apply Factor 1 (In the office example to buy a secondary screen for each worker, for instance).

For the minimization part Factor 5 seems to be the most important, and by a long shot. Factors 2 and 3 also seem to improve the minimization. One would expect that the combination of these three factors would improve even more the minimization value, but the truth is they heavily harms it. So, if we wanted to minimize our answer variable we would only apply Factor 5 (In the factory example to change the scheduling policy to build each component of the screw).

4.6 - Conclusions

As we have seen Yates algorithm is a great way to experiment with different factors and choose to apply those that fits the best for our needs. Though this system seems to be flawless, there a few important concepts to note.

First, Yates is based in predicted values using a model which means there can be an important variability in the results and we could even end up picking those factors which in reality are not the best ones for our needs.

Second, we have to define in a very precise manner all the factors to analyse, both semantically and numerally, without measurement issues.

Third, a real experimental analysis seems to perform as an NP-complete problem that depends on the size of the data, so as bigger our problem is, the more experiments we will need to do, resulting in exponential time to receive a conclusion. Reducing the problem size is a solution we have exemplified in this assignment, though, in a real-world scenario detecting all the factors that do not provide any information about the variability of the data is a way tougher experience. Moreover, in result of reducing the problem size, we may be accidentally eliminating important and useful data.

All in all, Yates is a great way to introduce new possible scenarios to complex problems. But as usual with prediction techniques the only real way to test a if a solution is actually useful we have to test it empirically.