Design of experiment exercise

Benjamin Cathelineau

25/12/2021

First approach

The first approach is to assume that the underlying function is of the form $f(x1, x2, \ldots, x11) = x1 \times c1 + x2 \times c2 + \cdots + x1 \times c11 + intercept + noise = y$

In that case, the best approach seems to figure out each of the of coefficient $(c1, c2, \ldots, c11, intercept)$, individually.

To do that, my approach was to set to 1 the coefficient that I wanted to figure out and to 0 all the others. To figure out the intercept, I set all the coefficients to 0.

```
cat first_experiment.csv
```

```
## 0,0,0,0,0,0,0,0,0,0,0
## 1,0,0,0,0,0,0,0,0,0,0
## 0,1,0,0,0,0,0,0,0,0,0
## 0,0,1,0,0,0,0,0,0,0,0
## 0,0,0,1,0,0,0,0,0,0,0
## 0,0,0,0,1,0,0,0,0,0,0
## 0,0,0,0,0,1,0,0,0,0,0
## 0,0,0,0,0,0,1,0,0,0,0
## 0,0,0,0,0,0,0,1,0,0,0
## 0,0,0,0,0,0,0,0,1,0,0
## 0,0,0,0,0,0,0,0,0,1,0
## 0,0,0,0,0,0,0,0,0,0,1
```

The idea is to repeat this experiment multiple time to minimize the effect of *noise* which is assumed to be random and uniform. I ran the experiment 8 times on the website and saved the result as a csv.

```
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'

## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag

## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
df =read.csv(file = "first_experiment_result.csv")
```

First, we figure out the intercept by using the experiments with all input set to 0.

```
intercept_exp=df
for (name in colnames(df)){
  name= toString(name)
  if(name != "y" && name != "Date"){
    intercept_exp= intercept_exp %>% filter(intercept_exp[[name]]==0)
  }
}
intercept=mean(intercept_exp$y)
intercept
```

```
## [1] 1.015288
```

Then we figure each individual coefficient by using the intercept:

```
for (name in colnames(df)){
  name= toString(name)
  if(name != "y" && name != "Date"){
    coeff_exp= df %>% filter(df[[name]]==1)
    print(paste(name,": ",mean(coeff_exp$y)-intercept))
  }
}
```

```
## [1] "x1 :  -0.595411814400044"
## [1] "x2 :  -0.00150854449204041"
## [1] "x3 :  0.00134383113797831"
## [1] "x4 :  1.00055712403771"
## [1] "x5 :  0.000488142660988178"
## [1] "x6 :  0.00074128407704821"
## [1] "x7 :  -0.000876641022951707"
## [1] "x8 :  -0.00145325712797417"
## [1] "x9 :  -1.99949921112206"
## [1] "x10 :  -0.000384207733760578"
## [1] "x11 :  0.000777744619766985"
```

We can use a linear regression for a sanity check and we see that we obtain exactly the same result. This linear regression will also allow us to predict.

```
form <- as.formula(paste("y~", paste0("x", 1:11, collapse="+")))
reg=lm(form,data=df)
reg
```

```
##
## Call:
## lm(formula = form, data = df)
##
## Coefficients:
## (Intercept)           x1           x2           x3           x4           x5
##   1.0152882   -0.5954118   -0.0015085    0.0013438    1.0005571    0.0004881
##          x6           x7           x8           x9          x10          x11
##   0.0007413   -0.0008766   -0.0014533   -1.9994992   -0.0003842    0.0007777
```

Now that we have a linear model, we can generate experiments to test it. We simply generate 10 set of 11 random number that we output to a csv file, so that we can copy paste it to the website. We fix a seed to avoid loosing our value by running the code multiple time.

```
set.seed(1)
new_random_experiment=t(runif(n=11))
```

```r
for (n in  1:9){
  appendable=t(runif(n=11))
  new_random_experiment=rbind(new_random_experiment,appendable)
}
write.table(x=new_random_experiment,file = "first_experiment_test.csv",row.names = FALSE,col.names = FA
```

Before even going to the website, we will now try to use our model to predict y.

Here are our experiments:

```r
colnames(new_random_experiment) <- c("x1","x2","x3","x4","x5","x6","x7","x8","x9","x10","x11")
new_random_experiment
```

```
##               x1        x2         x3         x4         x5        x6        x7
##  [1,] 0.26550866 0.3721239 0.57285336 0.90820779 0.20168193 0.8983897 0.9446753
##  [2,] 0.17655675 0.6870228 0.38410372 0.76984142 0.49769924 0.7176185 0.9919061
##  [3,] 0.65167377 0.1255551 0.26722067 0.38611409 0.01339033 0.3823880 0.8696908
##  [4,] 0.18621760 0.8273733 0.66846674 0.79423986 0.10794363 0.7237109 0.4112744
##  [5,] 0.52971958 0.7893562 0.02333120 0.47723007 0.73231374 0.6927316 0.4776196
##  [6,] 0.09946616 0.3162717 0.51863426 0.66200508 0.40683019 0.9128759 0.2936034
##  [7,] 0.47854525 0.7663107 0.08424691 0.87532133 0.33907294 0.8394404 0.3466835
##  [8,] 0.38998954 0.7773207 0.96061800 0.43465948 0.71251468 0.3999944 0.3253522
##  [9,] 0.24548851 0.1433044 0.23962942 0.05893438 0.64228826 0.8762692 0.7789147
## [10,] 0.60493329 0.6547239 0.35319727 0.27026015 0.99268406 0.6334933 0.2132081
##              x8        x9        x10        x11
##  [1,] 0.6607978 0.6291140 0.06178627 0.20597457
##  [2,] 0.3800352 0.7774452 0.93470523 0.21214252
##  [3,] 0.3403490 0.4820801 0.59956583 0.49354131
##  [4,] 0.8209463 0.6470602 0.78293276 0.55303631
##  [5,] 0.8612095 0.4380971 0.24479728 0.07067905
##  [6,] 0.4590657 0.3323947 0.65087047 0.25801678
##  [7,] 0.3337749 0.4763512 0.89219834 0.86433947
##  [8,] 0.7570871 0.2026923 0.71112122 0.12169192
##  [9,] 0.7973088 0.4552745 0.41008408 0.81087024
## [10,] 0.1293723 0.4781180 0.92407447 0.59876097
```

Now let's get the prediction from our model :

```r
for (n in 1:10){
  print(predict(object = reg,newdata=as.list(new_random_experiment[n,])))
}
```

```
##         1
## 0.5073228
##         1
## 0.1245721
##          1
## 0.04904047
##         1
## 0.4041131
##         1
## 0.2994102
##         1
## 0.9539371
##         1
## 0.6529872
```

```
##        1
## 0.811901
##          1
## 0.01746822
##           1
## -0.03030531
```

Now we can try running the experiment on the website to see if we get similar results. We save the result as a csv file to keep it for later.

```
first_experiment_results=read.csv(file = "first_experiment_test_results.csv")
first_experiment_results
```

```
##                   Date         x1        x2         x3         x4         x5
## 1  2021-12-29-09:44:47 0.26550866 0.3721239 0.57285336 0.90820779 0.20168193
## 2  2021-12-29-09:44:48 0.17655675 0.6870228 0.38410372 0.76984142 0.49769924
## 3  2021-12-29-09:44:49 0.65167377 0.1255551 0.26722067 0.38611409 0.01339033
## 4  2021-12-29-09:44:50 0.18621760 0.8273733 0.66846674 0.79423986 0.10794363
## 5  2021-12-29-09:44:51 0.52971958 0.7893562 0.02333120 0.47723007 0.73231374
## 6  2021-12-29-09:44:51 0.09946616 0.3162717 0.51863426 0.66200508 0.40683019
## 7  2021-12-29-09:44:53 0.47854525 0.7663107 0.08424691 0.87532133 0.33907294
## 8  2021-12-29-09:44:54 0.38998954 0.7773207 0.96061800 0.43465948 0.71251468
## 9  2021-12-29-09:44:55 0.24548851 0.1433044 0.23962942 0.05893438 0.64228826
## 10 2021-12-29-09:44:56 0.60493329 0.6547239 0.35319727 0.27026015 0.99268406
##           x6        x7        x8        x9        x10        x11          y
## 1  0.8983897 0.9446753 0.6607978 0.6291140 0.06178627 0.20597457 0.53893656
## 2  0.7176185 0.9919061 0.3800352 0.7774452 0.93470523 0.21214252 0.05792859
## 3  0.3823880 0.8696908 0.3403490 0.4820801 0.59956583 0.49354131 1.75858588
## 4  0.7237109 0.4112744 0.8209463 0.6470602 0.78293276 0.55303631 0.48204202
## 5  0.6927316 0.4776196 0.8612095 0.4380971 0.24479728 0.07067905 1.31945078
## 6  0.9128759 0.2936034 0.4590657 0.3323947 0.65087047 0.25801678 0.98110610
## 7  0.8394404 0.3466835 0.3337749 0.4763512 0.89219834 0.86433947 1.41765069
## 8  0.3999944 0.3253522 0.7570871 0.2026923 0.71112122 0.12169192 1.32207435
## 9  0.8762692 0.7789147 0.7973088 0.4552745 0.41008408 0.81087024 0.25692749
## 10 0.6334933 0.2132081 0.1293723 0.4781180 0.92407447 0.59876097 1.47170471
```

We immediately see that most of our prediction are completely of the mark. We can compare them with the actual result, and make an average of the error:

```
compare_prediction <- function(amount_experiment,model,data,result){

error_sum=0
for (n in 1:amount_experiment){
  prediction=predict(object = model,newdata=as.list(data[n,]))
  error=prediction-result$y[n]
  error_sum=error_sum+  abs(error)
  print(paste("prediction: ",prediction, ", Actual value: ",result$y[n],", Error: ",error))

}
error_sum/10
}
```

```
compare_prediction(amount_experiment = 10,model = reg, data = new_random_experiment,result = first_expe
```

```
## [1] "prediction:  0.507322792350404 , Actual value:  0.538936564882502 , Error:  -0.0316137725320981
## [1] "prediction:  0.124572121198922 , Actual value:  0.0579285899614864 , Error:  0.0666435312374352
## [1] "prediction:  0.0490404708336144 , Actual value:  1.75858587841955 , Error:  -1.70954540758594"
```

```
## [1] "prediction:  0.404113090568489 , Actual value:  0.482042015052203 , Error:  -0.0779289244837144
## [1] "prediction:  0.299410217242801 , Actual value:  1.31945077968118 , Error:  -1.02004056243838"
## [1] "prediction:  0.953937070920315 , Actual value:  0.981106101423224 , Error:  -0.0271690305029091
## [1] "prediction:  0.652987183420856 , Actual value:  1.4176506916448 , Error:  -0.764663508223944"
## [1] "prediction:  0.811901005890084 , Actual value:  1.32207434957929 , Error:  -0.510173343689206"
## [1] "prediction:  0.017468217411889 , Actual value:  0.256927493123191 , Error:  -0.239459275711302"
## [1] "prediction:  -0.0303053141737816 , Actual value:  1.47170470565657 , Error:  -1.50201001983035"

##         1
## 0.5949247
```

This is not surprising as, I asked the professor *Arnaud Legrand* on Mattermost and he told me that, while this was a decent first approach, the underlying function was *not* actually of the form $f(x1, x2, \dots, x11) = x1 \times c1 + x2 \times c2 + \cdots + x1 \times c11 + intercept + noise = y$

Second approach

Eliminate some coefficient

In my opinion we can use the results of the first experiment for the coefficient that have no effect, especially if we remember that *Arnaud Legrand* gave us the hint that some coefficient (ie some inputs) don't really affect the output. In fact, from the original experiment, it seems that only 3 coefficient matters[1] $(x1, x4, x9)$

This is risky, because theses coefficient that seem to not change the output $(x2, x3, x5, x6, x7, x8, x10, x11)$ according to the first design, may actually change the output in a complex way that we do not yet see. Nonetheless this would greatly increase the efficiency of the experiment if the assumption is correct.

I am willing to take that risk.

First, instead of the completely non randomized design that I had before, I'm now going to use the *Latin Hyper Square* design presented in class. After a lengthy process to install the library, I just slightly modified the code to adapt it to the current problem.

```
library(DoE.wrapper);   set.seed(42);
```

```
## Le chargement a nécessité le package : FrF2

## Le chargement a nécessité le package : DoE.base

## Le chargement a nécessité le package : grid

## Le chargement a nécessité le package : conf.design

## Registered S3 method overwritten by 'DoE.base':
##   method           from
##   factorize.factor conf.design

##
## Attachement du package : 'DoE.base'

## Les objets suivants sont masqués depuis 'package:stats':
##
##     aov, lm

## L'objet suivant est masqué depuis 'package:graphics':
##
##     plot.design
```

[1]in the meaning that they change the output significantly, ie more that 10^{-2}

```
## L'objet suivant est masqué depuis 'package:base':
##
##     lengths

## Le chargement a nécessité le package : rsm
```

```
d5_HD = lhs.design( type= "maximin" , nruns= 100 ,nfactors= 11,
    seed= 42 , factor.names=list( x1=c(0,1),x2=c(0,0),x3=c(0,0),x4=c(0,1),x5=c(0,0),x6=c(0,0),x7=c(0,0)


d5_HD[is.na(d5_HD)] <- 0
write.table(x=d5_HD,file = "second_experiment.csv",row.names = FALSE,col.names = FALSE,sep = ",")
```

Now I run those experiment on the website and get the results:

```
library(dplyr)
df =read.csv(file = "second_experiment_results.csv")
```

```
newreg=lm(y~x1+x4+x9,data=df)
newreg
```

```
##
## Call:
## lm.default(formula = y ~ x1 + x4 + x9, data = df)
##
## Coefficients:
## (Intercept)           x1           x4           x9
##      1.4583       0.8377       0.7301      -2.3822
```

I then run the prediction and compare them against the actual value:

```
compare_prediction(amount_experiment = 10,model = newreg,data = new_random_experiment,result = first_ex
```

```
## [1] "prediction:  0.84511166933275 , Actual value:  0.538936564882502 , Error:  0.306175104450248"
## [1] "prediction:  0.316213516398497 , Actual value:  0.0579285899614864 , Error:  0.258284926437011"
## [1] "prediction:  1.13770171680073 , Actual value:  1.75858587841955 , Error:  -0.620884161618824"
## [1] "prediction:  0.652729064097538 , Actual value:  0.482042015052203 , Error:  0.170687049045335"
## [1] "prediction:  1.2068405439139 , Actual value:  1.31945077968118 , Error:  -0.112610235767283"
## [1] "prediction:  1.23312355290295 , Actual value:  0.981106101423224 , Error:  0.252017451479727"
## [1] "prediction:  1.36348398568296 , Actual value:  1.4176506916448 , Error:  -0.0541667059618383"
## [1] "prediction:  1.6194988312837 , Actual value:  1.32207434957929 , Error:  0.297424481704413"
## [1] "prediction:  0.622420694857796 , Actual value:  0.256927493123191 , Error:  0.365493201734605"
## [1] "prediction:  1.02340104915334 , Actual value:  1.47170470565657 , Error:  -0.448303656503229"

##         1
## 0.2886047
```

The average error is 0.28. This is isn't perfect but it's a start, and it's ≈ 2 times better than the previous approach.

Remove x4

For some reason, the average error is slightly lower when we remove $x4$.

```
newreg3=lm(y~x1+x9,data=df)
newreg3
```

```
##
## Call:
```

```
## lm.default(formula = y ~ x1 + x9, data = df)
##
## Coefficients:
## (Intercept)           x1           x9
##      1.8531       0.7986      -2.4025
```

```
compare_prediction(amount_experiment = 10,model = newreg3,data = new_random_experiment,result = first_e
```

```
## [1] "prediction:  0.553656511492916 , Actual value:  0.538936564882502 , Error:  0.014719946610414"
## [1] "prediction:  0.126249750040711 , Actual value:  0.0579285899614864 , Error:  0.0683211600792243
## [1] "prediction:  1.21531119552158 , Actual value:  1.75858587841955 , Error:  -0.543274682897968"
## [1] "prediction:  0.447216214143494 , Actual value:  0.482042015052203 , Error:  -0.0348258009087089
## [1] "prediction:  1.22358392996412 , Actual value:  1.31945077968118 , Error:  -0.0958668497170552"
## [1] "prediction:  1.13391977033709 , Actual value:  0.981106101423224 , Error:  0.152813668913865"
## [1] "prediction:  1.09080852076207 , Actual value:  1.4176506916448 , Error:  -0.326842170882734"
## [1] "prediction:  1.67755264339738 , Actual value:  1.32207434957929 , Error:  0.355478293818087"
## [1] "prediction:  0.955318624787866 , Actual value:  0.256927493123191 , Error:  0.698391131664675"
## [1] "prediction:  1.18750158939375 , Actual value:  1.47170470565657 , Error:  -0.284203116262822"
```

```
##         1
## 0.2574737
```

Try with all coefficients

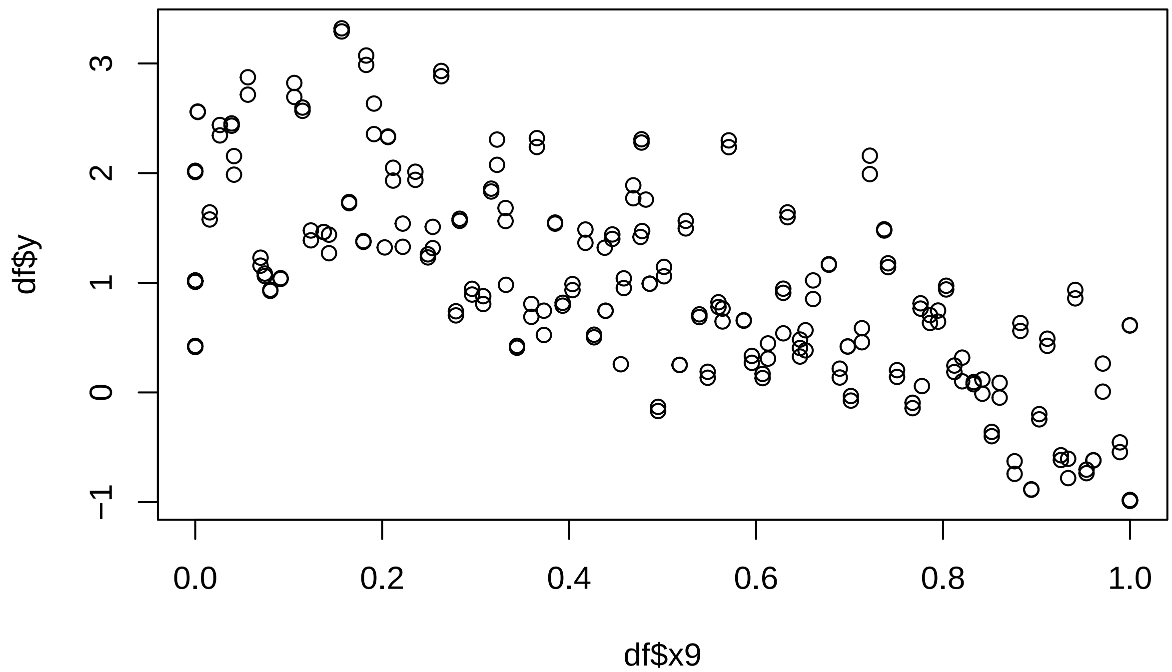
We redo a experiment design, this time with all the coefficient

```
library(DoE.wrapper);   set.seed(42);
d5_HD = lhs.design( type= "maximin" , nruns= 100 ,nfactors= 11,
    seed= 42 , factor.names=list( x1=c(0,1),x2=c(0,1),x3=c(0,1),x4=c(0,1),x5=c(0,1),x6=c(0,1),x7=c(0,1)


d5_HD[is.na(d5_HD)] <- 0
write.table(x=d5_HD,file = "third_experiment.csv",row.names = FALSE,col.names = FALSE,sep = ",")
```

```
library(dplyr)
df =read.csv(file = "third_experiment_results.csv")
```

```
plot(df$x9,df$y)
```




```
newreg2=lm(form,data=df)
newreg2
```

```
##
## Call:
## lm.default(formula = form, data = df)
##
## Coefficients:
## (Intercept)           x1           x2           x3           x4           x5
##     1.04849      0.69197      0.05872     -0.23648      1.04127      0.11842
##          x6           x7           x8           x9          x10          x11
##    -0.07546     -0.03209      0.02593     -1.95942      0.04781      0.12570
```

```
compare_prediction(amount_experiment = 10,model = newreg2,data = new_random_experiment,result = first_e
```

```
## [1] "prediction:  0.803341676877893 , Actual value:  0.538936564882502 , Error:  0.264405111995392"
## [1] "prediction:  0.452608183748029 , Actual value:  0.0579285899614864 , Error:  0.394679593786542"
## [1] "prediction:  0.945413139902232 , Actual value:  1.75858587841955 , Error:  -0.813172738517318"
## [1] "prediction:  0.700218399738827 , Actual value:  0.482042015052203 , Error:  0.218176384686624"
## [1] "prediction:  1.15641878443127 , Actual value:  1.31945077968118 , Error:  -0.163031995249912"
## [1] "prediction:  1.09659170145398 , Actual value:  0.981106101423224 , Error:  0.115485600030758"
## [1] "prediction:  1.50842671797187 , Actual value:  1.4176506916448 , Error:  0.090776026327068"
## [1] "prediction:  1.30493879807698 , Actual value:  1.32207434957929 , Error:  -0.0171355515023086"
## [1] "prediction:  0.466546695598546 , Actual value:  0.256927493123191 , Error:  0.209619202475355"
## [1] "prediction:  0.952294491378685 , Actual value:  1.47170470565657 , Error:  -0.519410214277885"
```

```
##         1
## 0.2805892
```

The average error is barely better than the $x1 + x4 + x9$ model, but worse than the $x1 + x9$ model…

Step reg model

I remembered the bat exercise and decided to try to use the `step` command.

```
reg0=lm(y~1,data = df)
stepreg= step(reg0,scope = form, direction = "forward")
```

```
## Start:  AIC=-98.47
## y ~ 1
##
##         Df Sum of Sq    RSS      AIC
## + x9     1    74.639 146.68 -222.751
## + x4     1    15.393 205.93 -118.596
## + x1     1     5.149 216.17 -103.691
## <none>               221.32  -98.465
## + x3     1     1.199 220.12  -98.133
## + x5     1     0.373 220.94  -96.983
## + x7     1     0.361 220.96  -96.966
## + x11    1     0.080 221.24  -96.577
## + x6     1     0.044 221.27  -96.526
## + x10    1     0.038 221.28  -96.518
## + x8     1     0.002 221.32  -96.468
## + x2     1     0.001 221.32  -96.466
##
## Step:  AIC=-222.75
## y ~ x9
##
##         Df Sum of Sq    RSS     AIC
## + x4     1    46.051 100.63 -336.43
## + x1     1    26.476 120.20 -281.86
## + x5     1     1.955 144.72 -224.87
## + x10    1     1.895 144.78 -224.74
## + x11    1     1.890 144.79 -224.73
## + x8     1     1.348 145.33 -223.58
## + x2     1     1.236 145.44 -223.35
## <none>               146.68 -222.75
## + x6     1     0.887 145.79 -222.61
## + x7     1     0.489 146.19 -221.78
## + x3     1     0.087 146.59 -220.93
##
## Step:  AIC=-336.43
## y ~ x9 + x4
##
##         Df Sum of Sq     RSS     AIC
## + x1     1   15.1952  85.432 -384.69
## + x11    1    0.6794  99.948 -336.51
## <none>               100.628 -336.43
## + x5     1    0.5833 100.044 -336.22
## + x2     1    0.2762 100.351 -335.28
## + x10    1    0.2452 100.382 -335.18
## + x3     1    0.1304 100.497 -334.83
## + x8     1    0.1156 100.512 -334.79
## + x7     1    0.0429 100.585 -334.57
```

```
## + x6    1    0.0352 100.592 -334.54
##
## Step:  AIC=-384.69
## y ~ x9 + x4 + x1
##
##        Df Sum of Sq    RSS     AIC
## <none>              85.432 -384.69
## + x3    1   0.40596 85.026 -384.15
## + x11   1   0.23295 85.199 -383.53
## + x5    1   0.17028 85.262 -383.30
## + x2    1   0.06358 85.369 -382.92
## + x10   1   0.05317 85.379 -382.88
## + x8    1   0.02377 85.409 -382.78
## + x6    1   0.01532 85.417 -382.75
## + x7    1   0.00045 85.432 -382.69
```

```
summary(stepreg)
```

```
##
## Call:
## lm.default(formula = y ~ x9 + x4 + x1, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.47850 -0.13429 -0.04331  0.13640  1.13419
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.05628    0.05410  19.523  < 2e-16 ***
## x9          -1.96958    0.09592 -20.534  < 2e-16 ***
## x4           1.04258    0.09388  11.105  < 2e-16 ***
## x1           0.69792    0.09507   7.341 1.96e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.531 on 303 degrees of freedom
## Multiple R-squared:  0.614,  Adjusted R-squared:  0.6102
## F-statistic: 160.6 on 3 and 303 DF,  p-value: < 2.2e-16
```

Notice that it does land on a $x9 + x4 + x1$.

Let's test this model:

```
compare_prediction(amount_experiment = 10,model = stepreg,data = new_random_experiment,result = first_e
```

```
## [1] "prediction:  0.949366005965204 , Actual value:  0.538936564882502 , Error:  0.410429441082702"
## [1] "prediction:  0.450877503797669 , Actual value:  0.0579285899614864 , Error:  0.392948913836183"
## [1] "prediction:  0.96415001566739 , Actual value:  1.75858587841955 , Error:  -0.79443586275216"
## [1] "prediction:  0.739860865333163 , Actual value:  0.482042015052203 , Error:  0.25781885028096"
## [1] "prediction:  1.06065943941762 , Actual value:  1.31945077968118 , Error:  -0.258791340263555"
## [1] "prediction:  1.16120952678661 , Actual value:  0.981106101423224 , Error:  0.180103425363388"
## [1] "prediction:  1.36463970940028 , Actual value:  1.4176506916448 , Error:  -0.0530109822445184"
## [1] "prediction:  1.38240497435419 , Actual value:  1.32207434957929 , Error:  0.0603306247749045"
## [1] "prediction:  0.392352681947729 , Actual value:  0.256927493123191 , Error:  0.135425188824538"
## [1] "prediction:  0.818546168651092 , Actual value:  1.47170470565657 , Error:  -0.653158537005478"
```

```
##         1
```

```
## 0.3196453
```

It's worse than the $x1 + x9$ model… :(.

step without x4

$x4$ is suspicious.

```r
reg0=lm(y~1,data = df)
stepreg2= step(reg0,scope = y~x1+x2+x3+x5+x6+x7+x8+x9+x10+x11, direction = "forward")
```

```
## Start:  AIC=-98.47
## y ~ 1
##
##         Df Sum of Sq    RSS     AIC
## + x9     1    74.639 146.68 -222.751
## + x1     1     5.149 216.17 -103.691
## <none>               221.32  -98.465
## + x3     1     1.199 220.12  -98.133
## + x5     1     0.373 220.94  -96.983
## + x7     1     0.361 220.96  -96.966
## + x11    1     0.080 221.24  -96.577
## + x6     1     0.044 221.27  -96.526
## + x10    1     0.038 221.28  -96.518
## + x8     1     0.002 221.32  -96.468
## + x2     1     0.001 221.32  -96.466
##
## Step:  AIC=-222.75
## y ~ x9
##
##         Df Sum of Sq    RSS     AIC
## + x1     1   26.4761 120.20 -281.86
## + x5     1    1.9546 144.72 -224.87
## + x10    1    1.8949 144.78 -224.74
## + x11    1    1.8905 144.79 -224.73
## + x8     1    1.3477 145.33 -223.59
## + x2     1    1.2355 145.44 -223.35
## <none>               146.68 -222.75
## + x6     1    0.8866 145.79 -222.61
## + x7     1    0.4893 146.19 -221.78
## + x3     1    0.0868 146.59 -220.93
##
## Step:  AIC=-281.86
## y ~ x9 + x1
##
##         Df Sum of Sq    RSS     AIC
## + x10    1   0.82672 119.38 -281.98
## <none>               120.20 -281.86
## + x11    1   0.73001 119.47 -281.73
## + x5     1   0.72937 119.47 -281.73
## + x8     1   0.64361 119.56 -281.51
## + x2     1   0.45391 119.75 -281.02
## + x6     1   0.18666 120.02 -280.34
## + x7     1   0.11069 120.09 -280.15
## + x3     1   0.02443 120.18 -279.93
```

```
##
## Step:  AIC=-281.98
## y ~ x9 + x1 + x10
##
##        Df Sum of Sq    RSS     AIC
## <none>              119.38 -281.98
## + x3    1   0.38011 119.00 -280.96
## + x11   1   0.25489 119.12 -280.64
## + x5    1   0.25245 119.12 -280.63
## + x8    1   0.19977 119.18 -280.50
## + x2    1   0.07980 119.30 -280.19
## + x7    1   0.00708 119.37 -280.00
## + x6    1   0.00006 119.38 -279.98
```

```
summary(stepreg2)
```

```
##
## Call:
## lm.default(formula = y ~ x9 + x1 + x10, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7288 -0.2421 -0.1158  0.3868  1.7193
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.25335    0.06151  20.375  < 2e-16 ***
## x9          -1.73684    0.11081 -15.673  < 2e-16 ***
## x1           0.88892    0.11069   8.031 2.17e-14 ***
## x10          0.16157    0.11153   1.449    0.148
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6277 on 303 degrees of freedom
## Multiple R-squared:  0.4606, Adjusted R-squared:  0.4553
## F-statistic: 86.25 on 3 and 303 DF,  p-value: < 2.2e-16
```

```
compare_prediction(amount_experiment = 10,model = stepreg2,data = new_random_experiment,result = first_
```

```
## [1] "prediction:  0.406670827707663 , Actual value:  0.538936564882502 , Error:  -0.132265737174839"
## [1] "prediction:  0.211006264068574 , Actual value:  0.0579285899614864 , Error:  0.153077674107087"
## [1] "prediction:  1.09220071278805 , Actual value:  1.75858587841955 , Error:  -0.666385165631499"
## [1] "prediction:  0.421531023624567 , Actual value:  0.482042015052203 , Error:  -0.0605109914276361
## [1] "prediction:  1.00286680211418 , Actual value:  1.31945077968118 , Error:  -0.316583977567005"
## [1] "prediction:  0.869604304517497 , Actual value:  0.981106101423224 , Error:  -0.111501796905726"
## [1] "prediction:  0.995533821454845 , Actual value:  1.4176506916448 , Error:  -0.422116870189955"
## [1] "prediction:  1.36286220683669 , Actual value:  1.32207434957929 , Error:  0.0407878572574025"
## [1] "prediction:  0.747079908341624 , Actual value:  0.256927493123191 , Error:  0.490152415218433"
## [1] "prediction:  1.10996352891427 , Actual value:  1.47170470565657 , Error:  -0.3617411767423"
```

```
##         1
## 0.2755124
```

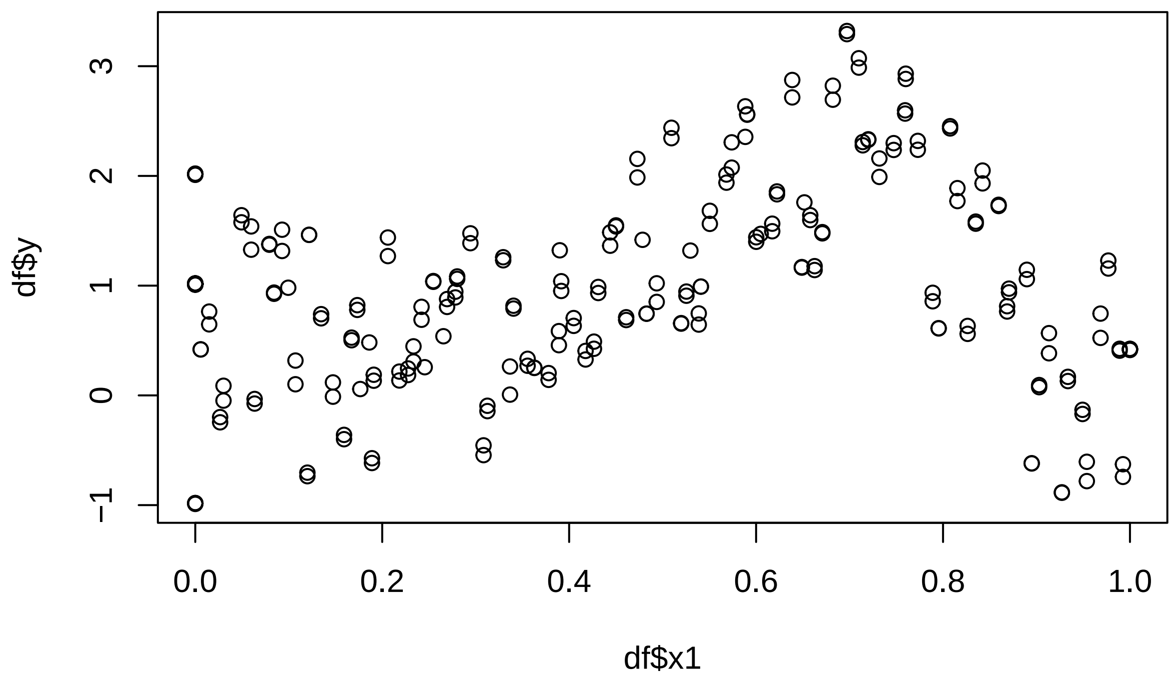
Slightly better…

Third approach : polynomial regression

Inspired by the hints of Arnaud Legrand, I guessed that the output was actually influenced by a polynomial of $x1$.

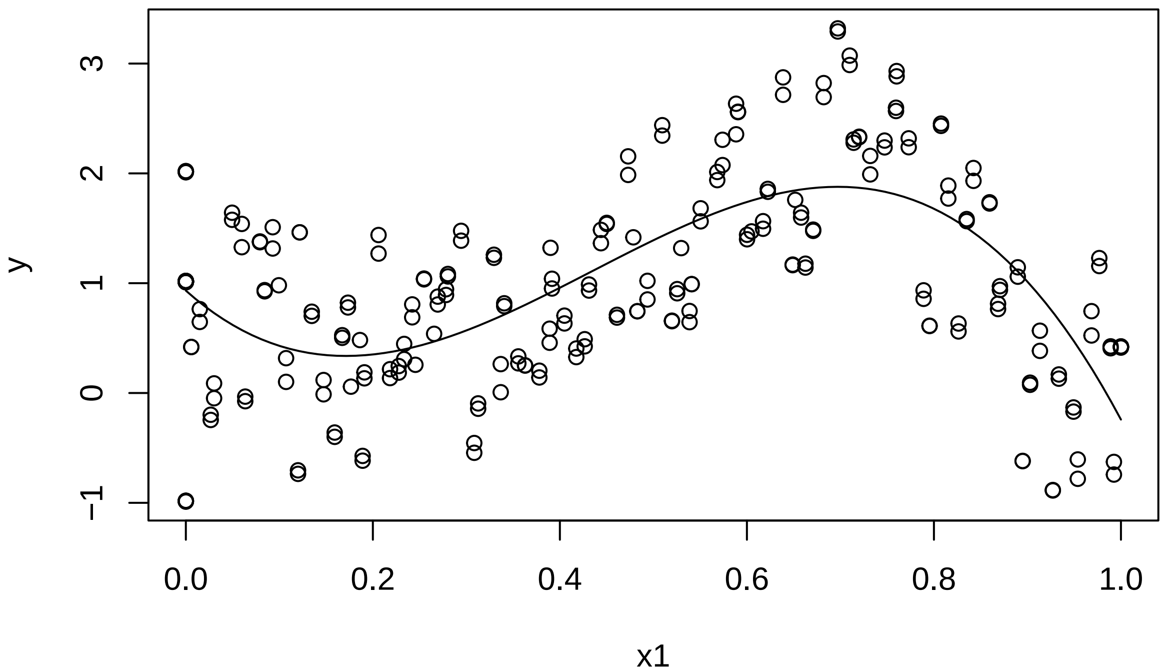
After testing different option i found that the best for $x1$ polynomial was $x1 + x1^2 + x1^3 =$:

```
x1polreg=lm(formula = y ~ x1 + I(x1^2) + I(x1^3),data = df)
plot(df$x1,df$y)
```




```
newdat = data.frame(x1 = seq(min(0), max(1), length.out = 100))
newdat$pred = predict(x1polreg, newdata = newdat)

plot(y ~ x1, data = df)
with(newdat, lines(x = x1, y = pred))
```



It looks very good graphically.

Now for the classical test that I did before.

```
polreg=lm( y ~ x1 + I(x1^2) + I(x1^3)+x4+x9,data = df)
```

```
compare_prediction(amount_experiment = 10,model = polreg,data = new_random_experiment,result = first_exp
```

```
## [1] "prediction:  0.637109172579445 , Actual value:  0.538936564882502 , Error:  0.0981726076969427"
## [1] "prediction:  0.00380995826488473 , Actual value:  0.0579285899614864 , Error:  -0.054118631696
## [1] "prediction:  1.72515476413456 , Actual value:  1.75858587841955 , Error:  -0.0334311142849868"
## [1] "prediction:  0.302290607164003 , Actual value:  0.482042015052203 , Error:  -0.1797514078882"
## [1] "prediction:  1.62157553633107 , Actual value:  1.31945077968118 , Error:  0.302124756649893"
## [1] "prediction:  0.771369813972774 , Actual value:  0.981106101423224 , Error:  -0.20973628745045"
## [1] "prediction:  1.74332492659961 , Actual value:  1.4176506916448 , Error:  0.325674234954814"
## [1] "prediction:  1.50117013414464 , Actual value:  1.32207434957929 , Error:  0.179095784565351"
## [1] "prediction:  0.103854914523552 , Actual value:  0.256927493123191 , Error:  -0.153072578599639"
## [1] "prediction:  1.54927993276014 , Actual value:  1.47170470565657 , Error:  0.0775752271035695"
```

```
##         1
## 0.1612753
```

This is a bit less than $\approx$ twice better (in term of average error) than the previously best model. Nice !

With the shiny app

With the shiny app I found an optimal value for $x1$ of around 0.74, $x4$ of around 1 and $x9$ of 0