Class 18 Demo: Generating Fractional Designs and Analyzing Experimental Data

```r
### Install packages
#install.packages("AlgDesign")

### Load packages:
#library(knitr)
library(janitor)
#library(readxl)
#library(psych)
#library(statar)
#library(tictoc)
#library(mktg482)
library(tidyverse)
#library(sjPlot)
#library(caret)
#library(glmnet)
#library(printr)
library(AlgDesign)
library(mktg482)

set.seed(3553)
```

Generating fractional designs

Generating a fractional design with the `AlgDesign` package is very straightforward. Let's start with the tools example. The first step is to create a list giving the factor names and levels. In design of experiments, it is typical to treat *all* variables as categorical; therefore, we always enter our factor levels in quotation marks (note: we do this even for numeric variables because this is equivalent to treating them numerically when there are only two levels and it would be inefficient to include more than two levels if we planned to treat them numerically).

```r
exp.design <- list( Threshold=c("200", "300"),
                    DiscountPct=c("15", "20"),
                    Coupon=c("Automatic", "Manual") )
exp.design
```

```
$Threshold
[1] "200" "300"

$DiscountPct
[1] "15" "20"

$Coupon
[1] "Automatic" "Manual"
```

Next, we generate the full factorial design using `expand.grid`:

```r
full.design <- expand.grid(exp.design)
full.design
```

```
  Threshold DiscountPct    Coupon
1       200          15 Automatic
2       300          15 Automatic
3       200          20 Automatic
4       300          20 Automatic
5       200          15    Manual
6       300          15    Manual
7       200          20    Manual
8       300          20    Manual
```

Finally, we generate the fractional design with the number of experimental condition we want to run using the `optFederov` function from the `AlgDesign` package. This function gives an optimal subset of conditions of size `nTrials` (here set to four) to run from the full design:

```r
frac.design <- optFederov(data=full.design, nTrials=4)
frac.design$design
```

```
  Threshold DiscountPct    Coupon
2       300          15 Automatic
3       200          20 Automatic
5       200          15    Manual
8       300          20    Manual
```

```r
frac.design$rows
```

```
[1] 2 3 5 8
```

Of course, there is not one single optimal choice so if we run this multiple times, we are likely to get different answers:

```r
optFederov(data=full.design, nTrials=4)$rows
```

```
[1] 1 4 6 7
```

```r
optFederov(data=full.design, nTrials=4)$rows
```

```
[1] 2 3 5 8
```

```r
optFederov(data=full.design, nTrials=4)$rows
```

```
[1] 1 4 6 7
```

```r
optFederov(data=full.design, nTrials=4)$rows
```

```
[1] 1 4 6 7
```

```r
optFederov(data=full.design, nTrials=4)$rows
```

```
[1] 1 4 6 7
```

Note: For a 2 x 2 x 2 design, if we want to run four trials, there are only two optimal choices. This is why we see the same two sets of rows repeated in the above.

Note: The minimum number of experimental conditions one can run is equal to one more than the sum of the number of levels of each experimental factor minus one, so here the smallest possible value for `nTrials` is 1 + (2-1) + (2-1) + (2-1) = 4. The maximum number of experimental conditions you can run is equal to the product of the number of levels of each experimental factor (i.e., the total number of unique experimental conditions), so here the largest possible value for `nTrials` is 2 x 2 x 2 = 8. Therefore, if we run the following code, we would get an error:

```
optFederov(data=full.design, nTrials=3)$rows
optFederov(data=full.design, nTrials=9)$rows
```

Obviously since this fractional factorial design contains the minimum number of experimental conditions, none of the interactions are estimable. We can confirm this with `get.estimable.coefficients`:

```
get.estimable.coefficients(full.design, frac.design)
```

```
$Estimable
[1] "Threshold300"  "DiscountPct20" "CouponManual"

$Inestimable
[1] "Threshold300:DiscountPct20"
[2] "Threshold300:CouponManual"
[3] "DiscountPct20:CouponManual"
[4] "Threshold300:DiscountPct20:CouponManual"
```

With the full design, obviously all are estimable. How about if we do a partial factorial with six conditions?

```
frac.design6 <- optFederov(data=full.design, nTrials=6)
get.estimable.coefficients(full.design, frac.design6)
```

```
$Estimable
[1] "Threshold300"               "DiscountPct20"
[3] "CouponManual"               "Threshold300:DiscountPct20"
[5] "Threshold300:CouponManual"

$Inestimable
[1] "DiscountPct20:CouponManual"
[2] "Threshold300:DiscountPct20:CouponManual"
```

We will now consider the Harrah's example, which followed a 3 x 3 x 2 x 3 x 2 x 2 experimental design. Suppose we want to run half (i.e., 108) of the 216 experimental conditions:

```
exp.design.harrahs <- list( Nights=c("0","1","2"),
                            Chips=c("0","50","100"),
                            Show=c("No","Yes"),
                            ExpDate=c("3","6", "12"),
                            Call=c("No","Yes"),
                            Email=c("No","Yes") )
full.design.harrahs <- expand.grid(exp.design.harrahs)
optFederov(data=full.design.harrahs, nTrials=108)$rows
```

```
  [1]   1   2   4   7   9  10  11  12  13  14  19  21  23  24  27  28  31
 [18]  34  35  38  39  41  49  51  52  53  54  57  61  62  66  67  68  69
 [35]  70  73  77  78  79  80  83  84  86  90  91  93  94  96  98  99 101
 [52] 104 107 108 110 114 116 117 119 123 125 126 130 131 133 134 135 136
 [69] 138 141 143 146 147 148 149 151 154 155 157 159 162 163 165 166 167
 [86] 168 171 173 175 178 179 181 186 188 190 191 192 194 195 196 199 200
[103] 203 205 210 211 214 216
```

```
optFederov(data=full.design.harrahs, nTrials=108)$rows
```

```
  [1]   2   6   7   8  10  13  16  17  18  20  21  23  26  29  30  35  36
 [18]  39  41  42  43  44  46  47  48  51  52  55  57  58  59  63  65  67
 [35]  68  72  73  75  76  78  80  82  85  87  88  89  91  94  95  99 100
 [52] 104 105 108 109 112 114 117 119 120 121 123 125 127 128 130 131 132
 [69] 133 136 140 142 144 145 149 151 153 156 157 158 159 164 165 167 169
 [86] 170 173 174 175 180 183 186 189 191 193 195 196 197 199 200 205 206
[103] 207 209 210 212 214 215
```

```
optFederov(data=full.design.harrahs, nTrials=108)$rows
```

```
  [1]   1   2   3   8  11  13  15  18  23  27  28  30  31  33  34  35  38
 [18]  40  41  42  43  44  45  46  47  50  52  55  57  58  59  61  64  66
 [35]  67  71  72  74  75  76  77  80  81  83  86  88  90  93  94  95 102
 [52] 105 106 108 110 113 114 115 117 118 119 123 124 125 127 128 129 133
 [69] 135 136 139 140 141 143 147 148 150 154 159 161 162 167 168 169 171
 [86] 174 175 176 179 181 182 184 186 187 192 195 197 199 200 205 206 207
[103] 208 209 210 211 212 215
```

As you can see, there are many possible choices.

Alternatively, suppose we want to run ten percent (i.e., ~22) of the 216 experimental conditions:

```
optFederov(data=full.design.harrahs, nTrials=22)$rows
```

```
 [1]   1   6  22  52  53  62  64  77  84  87  93 120 122 127 135 137 159
[18] 171 175 196 200 202
```

```
optFederov(data=full.design.harrahs, nTrials=22)$rows
```

```
 [1]   2  10  27  43  46  50  59  72  85  89  93 116 124 132 137 159 166
[18] 174 181 200 205 215
```

```
optFederov(data=full.design.harrahs, nTrials=22)$rows
```

```
 [1]  16  21  24  35  43  50  58  65  74  99 102 110 117 136 149 156 177
[18] 188 193 198 199 215
```

As you can see, there are many possible choices.

Obtaining desired interaction effects with fractional designs

Sometimes we may be interested in running a fractional design but, based on domain expertise, want to capture some interaction. For example, suppose in the tools example we had (correctly as it turns out) reasoned there was an interaction between the discount percentage and coupon entry, and we wanted to make sure we captured that. Because we were interested in this single pairwise interaction only, we thus would need to set `nTrials` to five (i.e., one more than the minimum of four). How can we make sure we get this interaction?

First, recall `optFederov` selects an optimal fractional design of size `nTrials` and that, when there are multiple such optimal fractional designs, it randomly chooses one. Given this, one can simply run `optFederov` followed by `get.estimable.coefficients` a few times to see if one obtains the desired interaction effect.

Second, after entering the required main effects, `get.estimable.coefficients` assumes you enter interactions in the model in the following manner: first, the pairwise interactions in the order of the variables as specified in the design matrix, then triple interactions again in the order of the variables as specified in the design matrix, and so on. That is, it assumes the model formula is of the form:

```
y ~ Factor1 * Factor2 * ... * Factork
```

Thus, in the tools example, since the design matrix puts `Threshold` first, `DiscountPct` second, and `Coupon` last, after entering the required main effects for each of these, it will next prioritize the pairwise interaction of `Threshold` and `DiscountPct`, then `Threshold` and `Coupon`, and then `DiscountPct` and `Coupon`; finally, it will enter the triple interaction of all three.

However, in your model formula, you need not follow this order. For instance, you could directly use as your model formula

```
y ~ Threshold + DiscountPct + Coupon + DiscountPct*Coupon
```

or, since the main effects have already been manually entered, equivalently

```
y ~ Threshold + DiscountPct + Coupon + DiscountPct:Coupon
```

which will estimate the main effects and also the desired interaction if it is estimable based on the fractional design.

Generally, you will want to know this before running the design! There are two basic ways to test this. Provided you follow standard practice and do not wish to "skip orders" of interactions (e.g., enter a three-way interaction but not all corresponding lower order two-way interactions), then when setting up the experimental design above, you can prioritize the variables as you see fit. For instance, in this case, we would instead write

```
exp.design2 <- list( DiscountPct=c("15", "20"),
                     Coupon=c("Automatic", "Manual"),
                     Threshold=c("200", "300") )
```

and use this in place of `exp.design` in the above. If we do this, `get.estimable.coefficients` will prioritize in the same fashion as you do.

Alternatively, whether or not you want to skip levels, you can also generate fake response data, run a model using your custom model formula, and see if the desired coefficient is estimated or is `NA`. For example, if we use the default formula it matches `get.estimable.coefficients`:

```
frac.design5 <- optFederov(data=full.design, nTrials=5)
get.estimable.coefficients(full.design, frac.design5)
```

```
$Estimable
[1] "Threshold300"                "DiscountPct20"
[3] "CouponManual"                "Threshold300:DiscountPct20"

$Inestimable
[1] "Threshold300:CouponManual"
[2] "DiscountPct20:CouponManual"
[3] "Threshold300:DiscountPct20:CouponManual"
```

```
fake.y <- rbinom(length(frac.design5$rows), 1, 0.5)      # Generate five binary variables with prob 1/2.
default.formula <- formula(fake.y ~ Threshold * DiscountPct * Coupon)
coef(glm(default.formula, family=binomial, data=frac.design5$design))
```

```
                            (Intercept) 
                           2.456607e+01 
                           Threshold300 
                           1.152953e-06 
                          DiscountPct20 
                          -3.287636e-15 
                           CouponManual 
                          -4.913214e+01 
             Threshold300:DiscountPct20 
                           4.913214e+01 
              Threshold300:CouponManual 
                                     NA 
             DiscountPct20:CouponManual 
                                     NA 
Threshold300:DiscountPct20:CouponManual 
```

```
                                          NA
```

But, if we use a custom formula, we get different interactions:

```
custom.formula <- formula(fake.y ~ Threshold + DiscountPct + Coupon + DiscountPct:Coupon)
coef(glm(custom.formula, family=binomial, data=frac.design5$design))
```

```
              (Intercept)               Threshold300
            -2.456607e+01               4.913214e+01
            DiscountPct20               CouponManual
             4.913214e+01               1.103271e-06
DiscountPct20:CouponManual
            -4.913214e+01
```

Indeed, in this case, we get our desired one!

Third, and finally, one can simply increase `nTrials` to obtain estimate more coefficients.

Note: The strategies outlined above, while perhaps klugey, are sufficient for this class. With more complicated experimental designs, more sophisticated and principaled strategies known to your data scientist are used in practice. I am happy to explain more in lab sessions, but it boils down to this. First, you figure out the `nTrials` required to estimate the coefficients you want (this is straightforward and is usually the number of coefficients plus one). Second, rather than having a procedure like `optFederov` select an optimal fractional design of that size (and when there are multiple such optimal fractional designs randomly choosing one), you instead have it select an optimal subject to the constraint that the desired coefficients are estimated. Unfortunately, `optFederov` does not have this capability internally at present (if you had had more programming experience, I could show you how to jury rig it to do so but it is complicated).

Analyzing experimental data

We typically get experimental data in summary form with rows denoting experimental conditions rather than individuals. Let me show you an example with the our original two group tools example. Recall, we had 580 sales out of 10000 exposures when the shipping threshold was set to \$200 and 500 sales out of 10000 exposures when the shipping threshold was set to \$300. Let's create a dataframe that reflects these results:

```
tools1.design <- list( Threshold=c("200", "300") )
tools1.full <- expand.grid(tools1.design)
tools1.full
```

```
  Threshold
1       200
2       300
```

```
tools1.results <- tools1.full %>%
  mutate(sales=c(580,500),
         fails=c(9420,9500))
tools1.results
```

```
  Threshold sales fails
1       200   580  9420
2       300   500  9500
```

How can we analyze summary data like this? We can use slightly different syntax for our logistic regression that weights the rows by the number of sales and fails respectively:

```
lr1 <- glm( cbind(sales,fails)~Threshold, data=tools1.results, family=binomial )
summary(lr1)
```

```
Call:
glm(formula = cbind(sales, fails) ~ Threshold, family = binomial,
    data = tools1.results)

Deviance Residuals:
[1]  0  0

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.78756    0.04278 -65.157   <2e-16 ***
Threshold300 -0.15688    0.06273  -2.501   0.0124 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance:  6.2696e+00  on 1  degrees of freedom
Residual deviance: -7.1498e-13  on 0  degrees of freedom
AIC: 20.143

Number of Fisher Scoring iterations: 2
```

We can also do this with a full factorial design. Consider, for example, our 2 x 2 full factorial design example:

```
tools2.design <- list( Threshold=c("200", "300"),
                       DiscountPct=c("15", "20") )
tools2.full <- expand.grid(tools2.design)
tools2.full
```

```
  Threshold DiscountPct
1       200          15
2       300          15
3       200          20
4       300          20
```

```
tools2.results <- tools2.full %>%
  mutate(sales=c(580,500,680,560),
         fails=c(9420,9500,9320,9440))
tools2.results
```

```
  Threshold DiscountPct sales fails
1       200          15   580  9420
2       300          15   500  9500
3       200          20   680  9320
4       300          20   560  9440
```

Next, we run a logistic regression with an interaction to capture the full factorial design:

```
lr2 <- glm( cbind(sales,fails)~Threshold*DiscountPct, data=tools2.results, family=binomial )
summary(lr2)
```

```
Call:
glm(formula = cbind(sales, fails) ~ Threshold * DiscountPct,
    family = binomial, data = tools2.results)

Deviance Residuals:
```

```
[1]  0  0  0  0

Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
(Intercept)                -2.78756    0.04278 -65.157  < 2e-16 ***
Threshold300               -0.15688    0.06273  -2.501  0.01240 *
DiscountPct20               0.16974    0.05838   2.907  0.00364 **
Threshold300:DiscountPct20 -0.05007    0.08605  -0.582  0.56065
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3.0391e+01  on 3  degrees of freedom
Residual deviance: 5.7288e-13  on 0  degrees of freedom
AIC: 40.541

Number of Fisher Scoring iterations: 2
```

Finally, we can do this with a fractional factorial design. Consider, for example:

```
tools3.design <- list( Threshold=c("200", "300"),
                    DiscountPct=c("15", "20"),
                    Coupon=c("Automatic", "Manual") )
tools3.full <- expand.grid(tools3.design)
tools3.full
```

```
  Threshold DiscountPct    Coupon
1       200          15 Automatic
2       300          15 Automatic
3       200          20 Automatic
4       300          20 Automatic
5       200          15    Manual
6       300          15    Manual
7       200          20    Manual
8       300          20    Manual
```

```
tools3.results <- tools3.full %>%
  mutate(sales=c(600,NA,NA,620,NA,500,640,NA),
         fails=c(9400,NA,NA,9380,NA,9500,9360,NA))
tools3.results
```

```
  Threshold DiscountPct    Coupon sales fails
1       200          15 Automatic   600  9400
2       300          15 Automatic    NA    NA
3       200          20 Automatic    NA    NA
4       300          20 Automatic   620  9380
5       200          15    Manual    NA    NA
6       300          15    Manual   500  9500
7       200          20    Manual   640  9360
8       300          20    Manual    NA    NA
```

Next, we run a logistic regression. We will include the interactions even though we already know we cannot estimate them (i.e., because in creating the fractional design, we assumed that they would be ignored). There is absolutely no harm in doing this: R will drop them because there is not enough information in the data to estimate them and, by including them, R will be sure to estimate any effects in the data that we actually can estimate!

```r
lr3 <- glm( cbind(sales,fails)~Threshold*DiscountPct*Coupon, data=tools3.results, family=binomial )
summary(lr3)
```

```

Call:
glm(formula = cbind(sales, fails) ~ Threshold * DiscountPct *
    Coupon, family = binomial, data = tools3.results)

Deviance Residuals:
[1]  0  0  0  0

Coefficients: (4 not defined because of singularities)
                                        Estimate Std. Error z value
(Intercept)                             -2.75154    0.04211 -65.345
Threshold300                            -0.11339    0.04262  -2.660
DiscountPct20                            0.14831    0.04262   3.480
CouponManual                            -0.07951    0.04262  -1.865
Threshold300:DiscountPct20                    NA         NA      NA
Threshold300:CouponManual                     NA         NA      NA
DiscountPct20:CouponManual                    NA         NA      NA
Threshold300:DiscountPct20:CouponManual       NA         NA      NA
                                        Pr(>|z|)
(Intercept)                              < 2e-16 ***
Threshold300                            0.007806 **
DiscountPct20                           0.000502 ***
CouponManual                            0.062126 .
Threshold300:DiscountPct20                    NA
Threshold300:CouponManual                     NA
DiscountPct20:CouponManual                    NA
Threshold300:DiscountPct20:CouponManual       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance:  2.1507e+01  on 3  degrees of freedom
Residual deviance: -5.7021e-12  on 0  degrees of freedom
  (4 observations deleted due to missingness)
AIC: 40.612

Number of Fisher Scoring iterations: 2
```

Note, this yields the same results as fitting a logistic regression without interactions and thus produces exactly the same predictions:

```r
lr3a <- glm( cbind(sales,fails)~Threshold+DiscountPct+Coupon, data=tools3.results, family=binomial )
summary(lr3a)
```

```

Call:
glm(formula = cbind(sales, fails) ~ Threshold + DiscountPct +
    Coupon, family = binomial, data = tools3.results)

Deviance Residuals:
[1]  0  0  0  0
```

```
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)   -2.75154    0.04211 -65.345  < 2e-16 ***
Threshold300  -0.11339    0.04262  -2.660 0.007806 **
DiscountPct20  0.14831    0.04262   3.480 0.000502 ***
CouponManual  -0.07951    0.04262  -1.865 0.062126 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance:  2.1507e+01  on 3  degrees of freedom
Residual deviance: -5.7021e-12  on 0  degrees of freedom
  (4 observations deleted due to missingness)
AIC: 40.612

Number of Fisher Scoring iterations: 2
```

```
predict(lr3, newdata=tools3.results, type="response")
```

```
         1          2          3          4          5          6 
0.06000000 0.05391474 0.06893134 0.06200000 0.05566941 0.05000000 
         7          8 
0.06400000 0.05753389 
```

```
predict(lr3a, newdata=tools3.results, type="response")
```

```
         1          2          3          4          5          6 
0.06000000 0.05391474 0.06893134 0.06200000 0.05566941 0.05000000 
         7          8 
0.06400000 0.05753389 
```

We can now use our logistic regression fit to the experimental conditions we did observe to extrapolate out to the ones we did not observe:

```
tools3.results <- tools3.results %>%
  mutate(pred = predict(lr3, newdata=tools3.results, type="response"),
         pred_sales = 10000*pred)
tools3.results
```

```
  Threshold DiscountPct    Coupon sales fails       pred pred_sales
1       200          15 Automatic   600  9400 0.06000000   600.0000
2       300          15 Automatic    NA    NA 0.05391474   539.1474
3       200          20 Automatic    NA    NA 0.06893134   689.3134
4       300          20 Automatic   620  9380 0.06200000   620.0000
5       200          15    Manual    NA    NA 0.05566941   556.6941
6       300          15    Manual   500  9500 0.05000000   500.0000
7       200          20    Manual   640  9360 0.06400000   640.0000
8       300          20    Manual    NA    NA 0.05753389   575.3389
```

This suggests condition three (which we did not run) may be the best one.