Practical Introduction to ExpertChoice

```r
library(ExpertChoice)
```

The purpose of this vignette is to present a practical worked example of how to design a discrete choice experiment without blocking. It follows two examples both explained at greater lengths in the complimentary vignette titled: “Theoretical introduction to `ExpertChoice`”

Step 0: Decide on what to test

The process of choosing a design often involves iterating over steps 0 to 4. Some designs are more difficult to create than others. The Theoretical Introduction to `ExpertChoice` presents two designs which are illustrated in this practical vignette.

Here are some practical suggestions as to what makes a good design. 1. In general avoid attributes with only two levels. The design such as the one below suffers because it is difficult to convert from the fractional factorial of this design into an efficient choice experiment design. The lack of efficiency is not from the methods of converting, but inherit in the fact that achieving the minimal overlap when there is only two levels is difficult.

Step 1: Construct the full factorial

First load the `ExpertDesign` package into your R environment.

```r
library(ExpertChoice)
```

Create a list object which specifies the name of the variables as well as their respective levels. **NB: In ordinal data the levels should be integer sequential and start from 0**. **NB: In categorical data the levels should be integer sequential and start from 1**. In an ordinal data experiment the level at 0 is always the reference point. As the proposed design above is a 5^5 I have chosen to denote the object as `attr55`:

```r
attri55 <- list(
  maker = c("0", "1", "2", "3", "4"),
  technical = c("0", "1", "2", "3", "4"),
  category_rarity = c("0", "1", "2", "3", "4"),
  size = c("0", "1", "2", "3", "4"),
  age = c("0", "1", "2", "3", "4")
)
```

Calling the list object something like this is advantageous because you could have multiple competing designs still at this stage. The following design is $4^4 2^1$ and then denoted here as `attri4521`:

```r
attri4521 <- list(
  maker = c("0", "1", "2", "3"),
  technical = c("0", "1", "2", "3"),
  category_rarity = c("0", "1", "2", "3"),
  size = c("0", "1", "2", "3"),
  age = c("0", "1", "2", "3"),
  provenance = c("0", "1")
)
```

Create the full factorial object. Using the design specification as a suffix remains a handy way of keeping track of the design.

```
ff55 <- full_factorial(attri55)
```

The full factorial will contain many rows. The first five rows and the last five are given below:

```
rbind(head(ff55, 5), tail(ff55, 5))
#>      maker technical category_rarity size age
#> 1        0         0               0    0   0
#> 2        1         0               0    0   0
#> 3        2         0               0    0   0
#> 4        3         0               0    0   0
#> 5        4         0               0    0   0
#> 3121     0         4               4    4   4
#> 3122     1         4               4    4   4
#> 3123     2         4               4    4   4
#> 3124     3         4               4    4   4
#> 3125     4         4               4    4   4
```

Every variable in the full factorial has the standardised orthogonal contrast applied. These contrasts are very useful when evaluating the efficacy of a design. This is simply illustrative of what the contrasts look like for one of the variables:

```
contrasts(ff55$maker)
#>        [,1]       [,2]       [,3] [,4]
#> 0  1.581139 -0.9128709 -0.6454972 -0.5
#> 1  0.000000  1.8257419 -0.6454972 -0.5
#> 2  0.000000  0.0000000  1.9364917 -0.5
#> 3  0.000000  0.0000000  0.0000000  2.0
#> 4 -1.581139 -0.9128709 -0.6454972 -0.5
```

Step 2: Augment the full factorial

Once the full factorial is constructed it is possible to augment it with additional information. Many of these augmentations happen as attributes. This includes adding the B-matrix for main effects, an important matrix in DCE efficiency, as described by Street et al… The prefix `aff` is used to refer to the augmented (full) factorial. (You could of course name the object whatever you prefer.)

```
aff55 <- augment_levels(ff55)
#> Applying B mat
```

A console log will appear stating that the processes of applying the B-matrix has started. If you do not get this message then the B-matrix cannot be added and a warning will be given. (Please open a

GitHub issue if this is the case. I am not aware of instances where this should happen.) The B-matrix plays an important role in the choice efficiency of design. Below are ten random rows drawn from the augmented full factorial. Notice the additional of the `levels` column.

```
aff55[sample(nrow(aff55), 10), ]
#>      maker technical category_rarity size age levels
#> 298      2         4               1    2   0  24120
#> 2192     1         3               2    2   3  13223
#> 846      0         4               3    1   1  04311
#> 2561     0         2               2    0   4  02204
#> 790      4         2               1    1   1  42111
#> 140      4         2               0    1   0  42010
#> 610      4         1               4    4   0  41440
#> 837      1         2               3    1   1  12311
#> 189      3         2               2    1   0  32210
#> 1717     1         3               3    3   2  13332
```

Step 3: Creating a fractional factorial design.

```
#library(AlgDesign)
library(DoE.base)
# library(DoE.MIParray)
```

There are many ways to create a fractional factorial design. Practically speaking though two methods are designed to be flawlessly integrated into this package. These are the construction of a fractional factorial design using an orthogonal array with either the `DoE.MIParray` or `DoE.base` packages or using D-optimal fractional factorial designs from the `AlgDesign` package. (The `AlgDesign` method could be better integrated. If you want to use this please open a GitHub issue. Will be sorted quickly.)

Orthogonal Arrays (`DoE.MIParray` or `DoE.base`)

Determine feasiability

The function `oa_feasible()` from the `DoE.base` package (`DoE.base::oa_feasible()`) provides many methods for determining if a particular design can be construed with N_D rows. For the silver research (by Jed Stephens) it was found that the following design was feasible. It is possible to specify higher resolution designs. There are reasons why higher resolution designs could be advantageous.

```
# Design: DF: 17, 32 OA (Resolution II), 64 OA (Resolution III)
nlevels <- unlist(purrr::map(ff55, function(x){length(levels(x))}))
oa_feasible(25, nlevels, strength = 2)
#> no violation of necessary criteria  for strength  2  was found
#> [1] TRUE
```

Using the `DoE.base` package it is possible to construct a 25 .

```r
fractional_factorial_55_25 <- oa.design(nlevels = nlevels, columns = "min34")
```

When constructing a design using the `DoE.MIParray`… The function `mosek_MIParray()` was used to construct the example 64 run orthogonal array included with this package.

```r
# Not run because it requires time as well as some setting up if this is your first
        time.
# See DoE.MIParray for more details.
#fractional_factorial_55_25 <- gurobi_MIParray(25, nlevels)
```

Note how the functions in DoE use a slightly different notation to index the factors. The base is given level 1 in this package. This is not really a concern because it will be seamlessly converted shortly.

```r
head(fractional_factorial_55_25, 10)
#>    A B C D E
#> 1  3 1 4 2 5
#> 2  5 1 2 3 4
#> 3  3 2 1 5 4
#> 4  4 1 3 5 2
#> 5  3 5 2 4 1
#> 6  4 4 4 4 4
#> 7  5 5 5 5 5
#> 8  1 4 2 5 3
#> 9  1 2 3 4 5
#> 10 3 3 3 3 3
```

D-efficient

Not yet discussed or though it should be achievable with minimal effort. If a reader wishes for this example to be completed before I have done so please open a GitHub issue and I shall happily oblige completing.

Step 4: Searching the full factorial for the chosen fractional factorial design

The ability to use multiple different packages to construct the fractional factorial design is ensured by this step. There can exist small differences between the different methods which require some fiddling.

The results of the `mosek_MIParray` function are orthogonal arrays without colnames. Hence in this instance the colnames need to be added. This design clearly needed to be made with the full factorial in mind. Hence the colnames from the `ff4521` object are appropriate. **Note: the colnames from the aff4521 would include the levels column – hence avoid these…**

```r
colnames(fractional_factorial_55_25) <- colnames(ff55)
fractional_f55_25 <- search_design(ff55, fractional_factorial_55_25)
```

The result is a fractional factorial design. Importantly though the fractional factorial design retains and inherits information from the full factorial such as the standardised orthogonal coding. To mark that many such attributes are held a special attribute is assigned to the object.

```
# Check to see if the searched attribute exists on the fractional_f4521_64 object.
attributes(fractional_f55_25)$searched
#> [1] TRUE
```

Once an object is search converted it is now easy to run diagnostics.

Step 5: Determining the efficacy of (full or fractional) factorial designs

The generalised world length patterns gives a good overall summary of the design.

```
DoE.base::GWLP(fractional_f55_25)
#>  0  1  2  3  4  5
#>  1  0  0 40 40 44
```

From this we can tell that this fractional factorial design is resolution IV i.e. strength of 3. Hence the all main effects are estimable free of each other, but some are confounded with two-attribute interactions.

The function `fractional_factorial_efficiency` provides a formula based method of investigating the proposed fractional factorial design in more details. This function also includes in its list of results the GWLP so there is no need to specify it.

Two examples are given which follow the two examples in the associated note.

```
# Test for main effects
main_effects <- fractional_factorial_efficiency(~ maker + technical +
        category_rarity + size + age, fractional_f55_25)
#> Your fractional factorial design has an A-efficiency of100%
#> Your fractional factorial design has a D-efficiency of100%
```

The resultant object has the following objects:

```
names(main_effects)
#> [1] "X"                  "information_mat"     "inv_information_mat"
#> [4] "lamda_mat"          "inv_diag"            "GWLP"
#> [7] "A_eff"              "D_eff"
```

Check the package help file for the `fractional_factorial_efficiency()` function for a full description. Also see the associated note for a more technical description.

```
# Test for main effects and interactions described in note.
#main_plus_interacts <- fractional_factorial_efficiency(~ maker * technical +
        category_rarity + size + age, fractional_f55_25)
```

This design supports only a single set of two-attribute interactions i.e. maker interact technical, or size interact age or age interact provenance etc. However it does not support more than two sets of two-attribute interactions: i.e. in this instance the maker interact technical and age interact provenance.

In instances where some of the stipulated effects cannot be estimated (such as above) then the D-efficiency would be NaN and similarly the A-efficiency is zero.

Step 6: Methods to convert from factorial designs to discrete choice experiments

In general there are a few methods for converting between fractional factorial to discrete choice experiments. The Theoretical introduction to `ExpertChoice` vignette gives more details. Although it is intended to implement the Modulo Method, L^{MA}, Rotation and mix-and-match, currently only the Modulo method is implemented. Which is sensible because as methods go it creates smaller and more efficient designs than these others. Notwithstanding this, if it is of interest to implement other methods these can be included in this package with limited difficulty. Please raise a GitHub issue.

Modulo Method

See the Theoretical introduction to `ExpertChoice` vignette for more details on how this method works.

The modulators are supplied as vectors contained within a list. The number of vectors provided determines the number of choice cards within a given choice set.

```
dce_modulo <- modulo_method(
  fractional_f55_25,
  list(c(1, 1, 1, 1, 1), c(3, 3, 3, 3, 3))
)
```

Step 7: Checking for Pareto Overshadowed Cards

Sometimes a particular choice may be Pareto dominate over all other choices. In such an instance there is no need to ask this question as it can be automatically answered and later augmented to the respondent data. It is important to remember to augment Pareto dominate choice sets if any exist.

```
checking_overshadow <- check_overshadow(dce_modulo)
# The matrix of indixes indicate that for row 1, 7, 13, 19 and 25 there are Pareto
        dominate solutions.
checking_overshadow
#>        [,1]  [,2]
#>  [1,]  TRUE  TRUE
#>  [2,] FALSE FALSE
#>  [3,] FALSE FALSE
#>  [4,] FALSE FALSE
#>  [5,] FALSE FALSE
#>  [6,] FALSE FALSE
#>  [7,]  TRUE  TRUE
#>  [8,] FALSE FALSE
#>  [9,] FALSE FALSE
#> [10,] FALSE FALSE
#> [11,] FALSE FALSE
#> [12,] FALSE FALSE
#> [13,]  TRUE  TRUE
#> [14,] FALSE FALSE
#> [15,] FALSE FALSE
#> [16,] FALSE FALSE
#> [17,] FALSE FALSE
#> [18,] FALSE FALSE
#> [19,]  TRUE  TRUE
#> [20,] FALSE FALSE
#> [21,] FALSE FALSE
#> [22,] FALSE FALSE
#> [23,] FALSE FALSE
#> [24,] FALSE FALSE
#> [25,]  TRUE  TRUE
```

Step 8: Efficacy of the Discrete Choice Design

This calculates the D-efficiency of the design as per Street, D.J., Burgess, L. and Louviere, J.J., 2005. Quick and easy choice sets: constructing optimal and nearly optimal stated choice experiments. International Journal of Research in Marketing, 22(4), pp.459-470. The pages of direct interest are 462 - 463 and these should be read in conjunction with the Theoretical introduction to `ExpertChoice` vignette. All the cases are printed and you can compare the calculation to the original paper.

```
dce_modulo_efficacy <- dce_efficiency(aff55, dce_modulo)
#>
#> q is1
#>   L is5
#> Case 4
#>   s is3
#>
#> q is2
#>   L is5
#> Case 4
#>   s is3
#>
#> q is3
#>   L is5
#> Case 4
#>   s is3
#>
#> q is4
#>   L is5
#> Case 4
#>   s is3
#>
#> q is5
#>   L is5
#> Case 4
#>   s is3
```

```
#>
#>
#> The D-efficiency of this discrete choice experiment is98.883%
```

Step 9: Construct a Discrete Choice Question Frame

The function `construct_question_frame` is helpful with the final stages. It consistently converts a `choice_set` arrangement into a `data.frame`.

```
question_table_f55 <- construct_question_frame(aff55, dce_modulo)
```

It is now time to add some useful information back to the levels. Originally these were described in Step 0, but up until this point it has been necessary to work with only integer values. (Also just imagine if you had worked with these very long names up until this point…)

```
levels(question_table_f55$maker) <- c("common (bottom 50% of makers)", "known to
        specialists (50% to 65% of makers)", "recognised (65% to 80%)", "famous (80%
        to 90%)", "celebrated top 10%")
levels(question_table_f55$technical) <- c("below average (below 50% of
        craftmanship)", "good (50% to 65%)", "meritorious (65% to 80%)",
        "distinguished (80% to 90%)", "exquisite (top 10%)")
levels(question_table_f55$category_rarity) <- c("common (bottom 20%)", "uncommon (20%
        to 40%)", "rare (40% to 60%)", "very rare (60% to 80%)", "exceptional (top
        20%)")
levels(question_table_f55$size) <- c("petite: under 125g", "small: between 126g and
        275g", "medium: between 276g and 600g", "large: between 601g and 1200g",
        "extra large: exceeds 1200g")
levels(question_table_f55$age) <- c("1951-present", "1951-present", "1951-present",
        "1951-present", "before 1800")
# View(question_table_f4521)
```

Viola! Here is the completed question dataframe.

```
question_table_f55
#> # A tibble: 75 x 8
#>    question choice levels maker     technical     category_rarity size    age
#>       <int>  <int> <chr>  <fct>     <fct>         <fct>           <fct>   <fct>
#>  1        1      1 11111  known to~ good (50% to~ uncommon (20% t~ small:~ 1951~
#>  2        1      2 33333  famous (~ distinguishe~ very rare (60% ~ large:~ 1951~
#>  3        1      3 00000  common (~ below averag~ common (bottom ~ petite~ 1951~
#>  4        2      1 33333  famous (~ distinguishe~ very rare (60% ~ large:~ 1951~
#>  5        2      2 00000  common (~ below averag~ common (bottom ~ petite~ 1951~
#>  6        2      3 22222  recognis~ meritorious ~ rare (40% to 60~ medium~ 1951~
#>  7        3      1 24130  recognis~ exquisite (t~ uncommon (20% t~ large:~ 1951~
#>  8        3      2 41302  celebrat~ good (50% to~ very rare (60% ~ petite~ 1951~
#>  9        3      3 13024  known to~ distinguishe~ common (bottom ~ medium~ befo~
#> 10        4      1 01234  common (~ good (50% to~ rare (40% to 60~ large:~ befo~
#> # ... with 65 more rows
```

The Restaurant Example

This DCE construction is described in the Theory Introduction to `ExpertChoice` vignette. It is intended to show more succinctly (than the silver example above) how to design such an experiment.

```
#Step 0
# Described in Theory
attri3261 <- list(
  starter = c("1", "2", "3"),
  main = c("1", "2", "3", "4", "5", "6"),
  dessert = c("1", "2", "3")
)
# Step 1
ff_examp <- full_factorial(attri3261)
# Step 2
aff_examp <- augment_levels(ff_examp)
#> Applying B mat
#write.csv(ff_examp, "example.csv")
# Step 3
nlevels <- unlist(purrr::map(ff_examp, function(x){length(levels(x))}))
#oa_feasible(36, nlevels, strength = 3)

fractional_factorial_3261_18 <- oa.design(nlevels = nlevels, columns = "min34")
# The fractional_factorial design is generated using the DoE.MIParray package.
# The following is the command to run this generation.
# The result is saved in the package.


# Step 4
# Confirming that this is an efficient design.
colnames(fractional_factorial_3261_18) <- colnames(ff_examp)
fractional_factorial_3261_18 <- search_design(ff_examp,
        fractional_factorial_3261_18)

# Step 5.
# This table is reported as Table
# Confirm that this design supports all interactions.
row1_main_effects <- fractional_factorial_efficiency(~ starter + main + dessert,
        fractional_factorial_3261_18)
#> Your fractional factorial design has an A-efficiency of100%
#> Your fractional factorial design has a D-efficiency of100%

# Step 6.
# Two different card options
# Option 1
dce_modulo_examp1 <- modulo_method(
  fractional_factorial_3261_18,
  list(c(1, 0, 1), c(0,1,0))
)

# Option 2.
dce_modulo_examp2 <- modulo_method(
  fractional_factorial_3261_18,
  list(c(1, 0, 1), c(1,3,1), c(0,5,0))
)

# Step 7
# This experiment uses categorical data (not ordinal) hence there can be no pareto
        dominate solution.
# Each category is merely a choice.


# Step 8.
# Compare the efficiencies.
dce_efficency_menu_example1 <- dce_efficiency(aff_examp, dce_modulo_examp1)
#>
#> q is1
#>   L is3
#>   Case 3
#>   The implied x is1and y is0
#>   s is3
#>
#> q is2
#>   L is6
#> Case 4
#>   s is3
#>
#> q is3
#>   L is3
#>   Case 3
#>   The implied x is1and y is0
#>   s is3
#>
#>
#> The D-efficiency of this discrete choice experiment is61.038%
dce_efficency_menu_example2 <- dce_efficiency(aff_examp, dce_modulo_examp2)
#>
#> q is1
#>   L is3
#>   Case 3
#>   The implied x is1and y is1
#>   s is5
#>
#> q is2
#>   L is6
#> Case 4
#>   s is6
#>
#> q is3
#>   L is3
#>   Case 3
#>   The implied x is1and y is1
#>   s is5
#>
#>
#> The D-efficiency of this discrete choice experiment is80.683%
```

```
# Option 2 is much more efficient so let's use that version!

# Step 9
# Construct the question table
menu_question_table <- construct_question_frame(aff_examp, dce_modulo_examp2)

# Finally augment the question table. See Table 1 in the Theoretical Vignette.
levels(menu_question_table$starter) <- c("Tomato Soup", "Duck Rillettes", "Seafood
        Chowder")
levels(menu_question_table$main) <- c("Roast Pheasant", "Pan Fried Hake", "Pork
        Belly", "Mushroom Risotto", "Sirloin Steak", "Vegetable Bake")
levels(menu_question_table$dessert) <- c("Sticky Toffee Pudding", "Chocolate &
        Hazelnut Brownie", "Cheesecake")
#View(menu_question_table)
```

Finally here is the resulting question table.

```
menu_question_table
#> # A tibble: 72 x 6
#>    question choice levels starter        main           dessert
#>       <int>  <int> <chr>  <fct>          <fct>          <fct>
#>  1        1      1 331    Seafood Chowd~ Pork Belly     Sticky Toffee Pudding
#>  2        1      2 132    Tomato Soup    Pork Belly     Chocolate & Hazelnut Br~
#>  3        1      3 162    Tomato Soup    Vegetable Bake Chocolate & Hazelnut Br~
#>  4        1      4 321    Seafood Chowd~ Pan Fried Hake Sticky Toffee Pudding
#>  5        2      1 341    Seafood Chowd~ Mushroom Riso~ Sticky Toffee Pudding
#>  6        2      2 142    Tomato Soup    Mushroom Riso~ Chocolate & Hazelnut Br~
#>  7        2      3 112    Tomato Soup    Roast Pheasant Chocolate & Hazelnut Br~
#>  8        2      4 331    Seafood Chowd~ Pork Belly     Sticky Toffee Pudding
#>  9        3      1 163    Tomato Soup    Vegetable Bake Cheesecake
#> 10        3      2 261    Duck Rillettes Vegetable Bake Sticky Toffee Pudding
#> # ... with 62 more rows
```

Replicating the example in Street *et al.*, 2005

The purpose of this section is to replicate the running example in Street, D.J., Burgess, L. and Louviere, J.J., 2005. Quick and easy choice sets: constructing optimal and nearly optimal stated choice experiments. International Journal of Research in Marketing, 22(4), pp.459-470.

The steps commented in the code follow those of the tutorial.

```
# Step 0
atttravel <- list(
  airfaire = c("0", "1"),
  travel_time = c("0", "1", "2")
)
# Step 1
travel2131 <- full_factorial(atttravel)
# Step 2
aff_travel2131 <- augment_levels(travel2131)
```

```r
#> Applying B mat
# Step 3.
# The full factorial is already so small that selecting a fraction of it would be
        silly.
# Therefore re-use the full factorial as the fractional factorial.

# Step 4.
# Confirming that this is an efficient design.
fractional_travel2131 <- search_design(travel2131, travel2131)

# Step 5.
# Confirm that this design supports all interactions.
full_factorial_efficiacy <- fractional_factorial_efficiency(~ (airfaire +
        travel_time)^2, fractional_travel2131)
#> Your fractional factorial design has an A-efficiency of100%
#> Your fractional factorial design has a D-efficiency of100%

# Step 6 & Step 7.
# Street gives two examples of choice sets.
travel_choice_set1 <- list(c("00", "11", "02"), c("10", "02", "12"))
class(travel_choice_set1) <- c(class(travel_choice_set1), "choice_set")

travel_example <- dce_efficiency(aff_travel2131, travel_choice_set1)
#>
#> q is1
#>   L is2
#>   Case 1
#>   s is2
#>
#> q is2
#>   L is3
#>   Case 3
#>   The implied x is1and y is0
#>   s is3
#>
#>
#> The D-efficiency of this discrete choice experiment is62.996%
# Note, if you want to rearrange the columns of the lamda matrix so that they are the
        same as Street use the following:
# lamda_street_cols  <- matrix(c(travel_example$Lamda$mat[,1],
#                                  travel_example$Lamda$mat[,3],
#                                  travel_example$Lamda$mat[,5],
#                                  travel_example$Lamda$mat[,2],
#                                  travel_example$Lamda$mat[,4],
#                                  travel_example$Lamda$mat[,6]), ncol = 6)
# lamda_street_paper <- matrix(c(lamda_street_cols[1,],
#                                  lamda_street_cols[3,],
#                                  lamda_street_cols[5,],
#                                  lamda_street_cols[2,],
#                                  lamda_street_cols[4,],
#                                  lamda_street_cols[6,]), ncol = 6)
# Street gives a second arrangement:
travel_choice_set2 <- list(c("00", "11", "02"), c("10", "01", "12"))
class(travel_choice_set2) <- c(class(travel_choice_set2), "choice_set")
# This version is 100% efficient.
```

```r
travel_example2 <- dce_efficiency(aff_travel2131, travel_choice_set2)
#> 
#> q is1
#>   L is2
#>   Case 1
#>   s is2
#> 
#> q is2
#>   L is3
#>   Case 3
#>   The implied x is1and y is0
#>   s is3
#> 
#> 
#> The D-efficiency of this discrete choice experiment is100%

# Step 8
travel_questions <- construct_question_frame(aff_travel2131, travel_choice_set2,
        randomise_choice_sets = FALSE)
levels(travel_questions$airfaire) <- c("$350", "$650")
levels(travel_questions$travel_time) <- c("4 hours", "5 hours", "6 hours")
#View(travel_questions)
```

References

Street, D.J., Burgess, L. and Louviere, J.J., 2005. Quick and easy choice sets: constructing optimal and nearly optimal stated choice experiments. International Journal of Research in Marketing, 22(4), pp.459-470.