Blocks Function Vignette

Benjamin Nguyen

November 20, 2018

Table of contents

What does blocks() do?

blocks() is a function I wrote to generate anaylsis for 2^k designs with confounding factors. Functionally, it generates a confounding set given an initial set of defining contrasts, a table that may be outputted which shows the 'guts' of the computation, and assigns treatments to one of the 2^q existing blocks for q defining contrasts. Furthermore, it converts a binary sequence into treatment letters, which I thought was a feature that was lacking in the conf.design package; it can become unwieldly to parse through the output of the conf.design package by eye when we have many factors (say $k \geq 5$), or if the factorial analysis is for $p \geq 3$, e.g. the factors have more than 2 levels.

The blocks function takes as inputs the defining contrasts (with some other sanity check parameters) and finds the parity (even or odd/ + or -) of the confounded variables based on a generated list of treatments, enumerated by all possible high/low combinations of k factors. It concatenates the parity of all the defining contrasts into a string (e.g. eee, eoe which is the same as +/+/+ and +/-/+ respectively) and assigns treatments to blocks based on if the treatment is associated with an existing parity sequence (i.e. ee, oe, eo, oo, for e := even, and o:=odd); it is not necessary to find sequences of length 3, which would include the generalized interaction of the 2 defining contrasts since it is not independent of the defining contrasts. More generally, we do not need to compute the parity of generalized interactions since they provide redundant information relative to a set of independent defining contrasts.

What can blocks() be used for?

There are many use cases for blocks(), and many possible extensions to make it more user-friendly and robust. It was generated for my personal use, so I forgoed some of these luxuries due to an increase in development time. Nonetheless, I have indexed some of the possible use cases for my function.

Example 1: Forming q blocks of size 2^{k-q}

In this example, suppose we are working with a 2^5 design, and we would like to form 4 blocks of size 8. We set the confounded variables to I = BCD = ACD. The function takes parameters k = number of factors, q = number of confounded variables in the defining relation I, and x = vector of length q containing the confounded variables as a binary sequence.

```r
# 2^5 factorial into 4 blocks of eight
# Confounders are BCD and ACD
ex1 <- blocks(k = 5, q = 2, c("01110", "10110"), table = TRUE, printset = TRUE)
```

```
## The confounded set is
## { BCD ACD AB }
##    treatment binary trt BCD ACD blocknumber
## 1        (1)      00000   e   e           1
## 2          a      10000   e   o           3
## 3          b      01000   o   e           2
## 4         ab      11000   o   o           4
## 5          c      00100   o   o           4
## 6         ac      10100   o   e           2
## 7         bc      01100   e   o           3
## 8        abc      11100   e   e           1
## 9          d      00010   o   o           4
## 10        ad      10010   o   e           2
## 11        bd      01010   e   o           3
## 12       abd      11010   e   e           1
## 13        cd      00110   e   e           1
## 14       acd      10110   e   o           3
## 15       bcd      01110   o   e           2
## 16      abcd      11110   o   o           4
## 17         e      00001   e   e           1
## 18        ae      10001   e   o           3
## 19        be      01001   o   e           2
## 20       abe      11001   o   o           4
## 21        ce      00101   o   o           4
## 22       ace      10101   o   e           2
## 23       bce      01101   e   o           3
## 24      abce      11101   e   e           1
## 25        de      00011   o   o           4
## 26       ade      10011   o   e           2
## 27       bde      01011   e   o           3
## 28      abde      11011   e   e           1
## 29       cde      00111   e   e           1
## 30      acde      10111   e   o           3
## 31      bcde      01111   o   e           2
## 32     abcde      11111   o   o           4
```

```r
# The assignment of treatments into 4 blocks is given below.
```

The above output generates a list of all treatments expressed as a sequence of letters or a sequence of binary numbers in column 1 and 2 respectively. Columns 4 and 5 are associated with the confounded variables in the defining relation; it finds the parity of the treatment in the corresponding row and evalues it for parity (even or odd). I assign a block number that can uniquely identify the concatenated sequence of parity values across the columns of confounded variables.

The following output can be condensed to show that we have indeed formed 4 blocks of size 8.

```r
ex1
```

```
##   eeblock oeblock eoblock ooblock
## 1     (1)       b       a      ab
## 2     abc      ac      bc       c
## 3     abd      ad      bd       d
```

```
## 4      cd     bcd     acd    abcd
## 5       e      be      ae     abe
## 6    abce     ace     bce      ce
## 7    abde     ade     bde      de
## 8     cde    bcde    acde   abcde
```

Example 2: Generating all the aliases of the defining relation

The blocks function internally uses the conf.set function in the package conf.design and converts the output into a concatenated binary sequence that is converted to a string that is then outputted as treatment combinations; absence of a letter indicates the low level; inclusion of a (lowercase) letter indicates the high level of a factor.

To show the benefit of using my function, we will use the function in conf.set to generate all the aliases of the defining relation.

```
gen = rbind(c(1,1,1,0,0,1,0,0),c(1,1,0,1,1,0,0,0),
            c(1,0,1,1,1,0,0,0), c(0,1,1,1,0,0,0,1))
cset = conf.set(gen, 2)
colnames(cset) = LETTERS[1:8]
cset
```

```
##       A B C D E F G H
##  [1,] 1 1 1 0 0 1 0 0
##  [2,] 1 1 0 1 1 0 0 0
##  [3,] 0 0 1 1 1 1 0 0
##  [4,] 1 0 1 1 1 0 0 0
##  [5,] 0 1 0 1 1 1 0 0
##  [6,] 0 1 1 0 0 0 0 0
##  [7,] 1 0 0 0 0 1 0 0
##  [8,] 0 1 1 1 0 0 0 1
##  [9,] 1 0 0 1 0 1 0 1
## [10,] 1 0 1 0 1 0 0 1
## [11,] 0 1 0 0 1 1 0 1
## [12,] 1 1 0 0 1 0 0 1
## [13,] 0 0 1 0 1 1 0 1
## [14,] 0 0 0 1 0 0 0 1
## [15,] 1 1 1 1 0 1 0 1
```

It can be a chore to identify the factors in the confounding set, since the output of conf.design is in binary sequence and the length of the sequence is quite lengthy. My function also generates the same confounding set, but it outputs it as treatment names, so we can identify them immediately.

```
# Defining the defining contrasts.
# 12345678
# abcdefgh
# 11100100
# 11011000
# 10111000
# 01110001
# Provided that I have defined
# the contrasts ABCF, ABDE, ACDE, and BCDH #
# without error, the confounded set is
ex2 = blocks(8, 4, c("11100100", "11011000",
                     "10111000", "01110001"), print = TRUE)
```

```
## The confounded set is
## { ABCF ABDE CDEF ACDE BDEF BC AF BCDH ADFH ACEH BEFH ABEH CEFH DH ABCDFH }
```

Example 3: Fractional Factorial Designs

Sometimes two series designs can contain many factors; far too many that can be used practically in an experimental setting. In these cases, we consider fractional designs.

For example, consider a 2^{6-2} fractional factorial using I = ABDF = -BCDE. $2^6 = 64$ treatments, which is quite a lot of treatments to implement; so instead we consider $2^4 = 16$ treatments which is much more manageable. We use defining relations of length 4 to minimize the chances of aliasing important effects.

For instance, in a 2^{6-2} design, we have 6 main effects that we want to estimate. We want to figure out how these main effects are aliased to make sure that none of them appear in our confounded set.

Given that I = ABDF = -BCDE, we find the generalized interaction as -ACEF and these form the confounded set.

We can find the aliases of the main affect by component wise addition modulo 2 on I in the presence of each main effect taken one at a time.

A$\odot$I = BDF = -ABCDE = -CEF

B$\odot$I = ADF = -CDE = -ABCEF

C$\odot$I = ABCDF = - BDE = -AEF

D$\odot$I = ABF = -BCE = -ACDEF

E$\odot$I = ABDEF = -BCD = -ACF

F$\odot$I = ABD = -BCDEF = -ACE

Suppose we want to find the factor-level combinations used in a 1/4 fractional factorial given the defining relation. This corresponds to 1) finding the appropriate block and 2) finding the elements of the block.

Using the blocks() function, I can pull the desired results by extrapolating the results of a confounded two-series design to a two-series fractional design. Confounded two-series designs are closely related to two-series fractional designs, so the key step to mapping the results of a confounded design to the fractional design is to follow the signs of the defining relation.

```
k = 6
q = 2
x = c("110101", "011110")
des1 = blocks(k,q,x,p=T, t = T)
```

```
## The confounded set is
## { ABDF BCDE ACEF }
##    treatment binary trt ABDF BCDE blocknumber
## 1        (1)     000000    e    e           1
## 2          a     100000    o    e           2
## 3          b     010000    o    o           4
## 4         ab     110000    e    o           3
## 5          c     001000    e    o           3
## 6         ac     101000    o    o           4
## 7         bc     011000    o    e           2
## 8        abc     111000    e    e           1
## 9          d     000100    o    o           4
## 10        ad     100100    e    o           3
## 11        bd     010100    e    e           1
```

```
## 12       abd     110100    o    e           2
## 13        cd     001100    o    e           2
## 14       acd     101100    e    e           1
## 15       bcd     011100    e    o           3
## 16      abcd     111100    o    o           4
## 17         e     000010    e    o           3
## 18        ae     100010    o    o           4
## 19        be     010010    o    e           2
## 20       abe     110010    e    e           1
## 21        ce     001010    e    e           1
## 22       ace     101010    o    e           2
## 23       bce     011010    o    o           4
## 24      abce     111010    e    o           3
## 25        de     000110    o    e           2
## 26       ade     100110    e    e           1
## 27       bde     010110    e    o           3
## 28      abde     110110    o    o           4
## 29       cde     001110    o    o           4
## 30      acde     101110    e    o           3
## 31      bcde     011110    e    e           1
## 32     abcde     111110    o    e           2
## 33         f     000001    o    e           2
## 34        af     100001    e    e           1
## 35        bf     010001    e    o           3
## 36       abf     110001    o    o           4
## 37        cf     001001    o    o           4
## 38       acf     101001    e    o           3
## 39       bcf     011001    e    e           1
## 40      abcf     111001    o    e           2
## 41        df     000101    e    o           3
## 42       adf     100101    o    o           4
## 43       bdf     010101    o    e           2
## 44      abdf     110101    e    e           1
## 45       cdf     001101    e    e           1
## 46      acdf     101101    o    e           2
## 47      bcdf     011101    o    o           4
## 48     abcdf     111101    e    o           3
## 49        ef     000011    o    o           4
## 50       aef     100011    e    o           3
## 51       bef     010011    e    e           1
## 52      abef     110011    o    e           2
## 53       cef     001011    o    e           2
## 54      acef     101011    e    e           1
## 55      bcef     011011    e    o           3
## 56     abcef     111011    o    o           4
## 57       def     000111    e    e           1
## 58      adef     100111    o    e           2
## 59      bdef     010111    o    o           4
## 60     abdef     110111    e    o           3
## 61      cdef     001111    e    o           3
## 62     acdef     101111    o    o           4
## 63     bcdef     011111    o    e           2
## 64    abcdef     111111    e    e           1
```

```
des1
```

```
##    eeblock oeblock eoblock ooblock
## 1      (1)       a      ab       b
## 2      abc      bc       c      ac
## 3       bd     abd      ad       d
## 4      acd      cd     bcd    abcd
## 5      abe      be       e      ae
## 6       ce     ace    abce     bce
## 7      ade      de     bde    abde
## 8     bcde   abcde    acde     cde
## 9       af       f      bf     abf
## 10     bcf    abcf     acf      cf
## 11    abdf     bdf      df     adf
## 12     cdf    acdf   abcdf    bcdf
## 13     bef    abef     aef      ef
## 14    acef     cef    bcef   abcef
## 15     def    adef   abdef    bdef
## 16  abcdef   bcdef    cdef   acdef
```

First, the defining contrasts are inputted for ABDF, and BCDE for the confounded design. In the defining relation, we want ABCF and -BCDE. When constructing a design matrix, we know that ABCF will be positive when there are an *even* number of factors at the high level, when we look at the subset of factors A, B, C, F. Similarly, we know that BCDE will be negative when there are an *odd* number of factors at the high level, when looking at the subset of factors B, C, D, E. The design matrix can be constructed using component-wise addition modulo 2 on the exponenets of the factors, or it can be done in the usual methods of constructing a table of contrasts (by enumerating all high-low combinations of the present factors, then looking at their products to generate the desired contrast for the higher-order interaction).

The defining relation has $I = W_1 = -W_2 = -W_1W_2$. For word 1 = ABDF and word 2 = -BCDE, the block that is chosen by this defining relation is precisely the e/o block (for the reasons I stated above in regards to the number of required high level factors to generate the appropriate sign).

The e/o block contains the elements ab, c, ad, bcd, e, abce, bde, acde, bf, acf, df, abcdf, aef, bcef, abdef, and cdef. These elements make up the treatments that will be used in a run in a 1/4 fractional factorial design.

Suppose that we would like to block these combinations into two blocks of size eight.

To partition this block further into two blocks of size 8, we need to select an effect to be confounded with the between-block differences generated by partitioning our e/o block into two blocks.

A simple choice is to select the AB interaction to be confounded with block differences. From this decision, I look at the elements of e/o; for each element I inspect if they have an even or odd number of A and B factors at the high level (this is done by counting the numbers of a, b that occur in our treatments, as the presence of a, b denote the presence of A, B at their high level). Based on the count of even or odd number of high levels A and B for each treatment, I deposit the treatment into and even or an odd block.

Keeping in mind that we are in a $2^{6-2} = 2^4$ design, the sample space for counts is $\{0, 1, 2, 3, 4\}$, and the even block consists of counts $\{0, 2, 4\}$, and the odd block consists of counts $\{1, 3\}$ for the number of factors at the high level of A, B.

This procedure generates the two blocks of size 8:

Even block: ab, c, e, abce, df, abcdf, abdef, cdef

Odd block: ad, bcd, bde, acde, bf, acf, aef, bcef

More generally, we can choose any other factor combinations to confound with blocks, so long as we choose factor combinations that exist in the e/o block.

Example 4: Generating all aliases of an effect

Consider the 2^{8-4} fractional factorial with generator I = BCDE = ACDF = ABCG = ABDH. Find the aliases of C.

The trick to this problem is that we have 4 words in our defining relation. This means that there are going to be a total of 15 words that are aliases of I (4 choose 1 + 4 choose 2 + 4 choose 3 + 4 choose 4).

Inputting the words into blocks() will spell out all the words that are aliases of I.

```
k = 8
q = 4
x = c("01111000", "10110100",
      "11100010", "11010001")
des2 = blocks(k,q,x,p=T)
```

```
## The confounded set is
## { BCDE ACDF ABEF ABCG ADEG BDFG CEFG ABDH ACEH BCFH DEFH CDGH BEGH AFGH ABCDEFGH }
```

Given the aliases of I, we can find the aliases of C by taking component wise addition modulo 2 on the exponents for c $\odot$ I.

c $\odot$ I = BDE = ADF = ABCEF = ABG = ACDEG = BCDFG = EFG = ABCDH = AEH = BFH = CDEFH = DGH = BCEGH = ACFGH = ABDEFGH.

Extensions/Alterations

As shown above, there are many use cases for the blocks function. In example 2, I used the conf.design function, but it could also be implemented on my end.

An alternative procedure that would generate the same output would be to take elements of the defining contrast and to do a special operation on them which would also generate the confounded set. The special operation would be element-wise addition modulo p over the exponents of factor combinations in the defining contrast set, where we take j elements at a time from the set of defining contrasts for j = 1, . . . , k.

E.g. For j = 1, we inspect the elements of the defining contrast set and find that all we do is deposit each element into the confounded set. For the defining contrast set ABCF, ABDE, ACDE, and BCDH, j = 1 would find these elements of the confounded set.

For j = 2, we take every possible combination for two elements, (q choose j == |defining set| choose j == # of defining contrasts choose j), and do our special operation and deposit the results into a confounded set. We repeat this procedure iteratively until we exhaustively inspect every number of possible selection (up until j = k, for k := number of factors).

Less concisely,

For j = 1,

ABCF

ABDE

ACDE

BCDH

For j = 2,

ABCF $\odot$ ABDE = CDEF

ABCF $\odot$ ACDE = BDEF,

ABCF $\odot$ BCDH = ADFH,

ABDE $\odot$ ACDE = BC,

ABCF $\odot$ BCDH = ADFH,

ACDE $\odot$ BCDH = ABEH,

and that specifies all q choose 2 generalized interaction.

Similarly, for j = 3,

ABCF $\odot$ ABDE $\odot$ ACDE $\rightarrow A^3B^2C^2D^2E^2F^1H^0 \xRightarrow[Mod2]{Exponents}$ AF,

ABCF $\odot$ ABDE $\odot$ BCDH = BEFH,

ABDE $\odot$ ACDE $\odot$ BCDH = DH,

ABCF $\odot$ ACDE $\odot$ BCDH = CEFH,

For j = 4 = k

ABCF $\odot$ ABDE $\odot$ ACDE $\odot$ BCDH = ABCDFH

and that enumerates all the confounded variables through an interative procedure (not using conf.set).

Mathematically, the confounded set can be defined as $f\left(\binom{q}{j}_{j=1,..k}\right)$, where f is a function that uses the binomial coefficient input and combinatorically exhausts all combinations of j factors taken at a time over the number of defining contrasts q and does the special operation mentioned above. As a sanity check $\sum_{j=1}^{k}\binom{q}{k}$ will generate a number that is identically equal to $df_{blocks} = 2^q - 1$; j = 1 enumerates all the defining contrasts and $j \neq 1$ generates all generalized interactions; that means that $\sum_{j=2}^{k}\binom{q}{k}$ will be equal to $df_{blocks} - q = p^q - 1 - q$ and that specifies the number of generalized interactions for a factor with p levels in a p^k design.

This procedure would have been implemented if the function from conf.design had not been readily available.

I could also generate an extension from example 4; we had to define the confounding set before we could define the aliases of the main effect of C. This would amount to component-wise modulo addition between two strings; one containing the confounding set and another containing the effect to de-alias.

I could also extend my function to handle factors with prime number of levels; this would require refactoring my code.

The number of use cases that my function contains can certainly be extended to a whole library of functions and be sorted into a package.