```
> des1 <- FrF2(16,5);des1
```

The FrF2 package has several functions for working with fractioned designs. The basic function is FrF2(). We can use it to construct designs using generators that we specify, or it can select a design for us when we specify number of runs and number of factors.
In the usage here, we want it to find a design for 5 factors in sixteen runs. The output is a data frame with the factor levels (in +1, –1 form). By default, the order of the runs will be randomized.

```
    A  B  C  D  E
1   1 -1 -1 -1 -1
2  -1  1 -1 -1 -1
3   1 -1  1  1 -1
4  -1 -1 -1 -1  1
5   1  1 -1 -1  1
6   1  1  1 -1 -1
7   1 -1  1 -1  1
8  -1  1  1  1 -1
9  -1 -1  1 -1 -1
10 -1  1  1 -1  1
11 -1 -1 -1  1 -1
12 -1  1 -1  1  1
13  1 -1 -1  1  1
14 -1 -1  1  1  1
15  1  1 -1  1 -1
16  1  1  1  1  1
class=design, type= FrF2
```

```
> FrF2(4,3,factor.names=c("Moe","Larry","Curly"))
```

You can supply factor names if you don't like A, B, C, etc.

```
  Moe Larry Curly
1  -1     1    -1
2   1     1     1
3   1    -1    -1
4  -1    -1     1
class=design, type= FrF2
```

```
> FrF2(4,3,factor.names=list(speed=c("fast","slow"),temp=c("hot","cold"),
    time=c("long","short")))
```

You can also name the levels by using a list with named components, each of which contains the level names.

```
  speed temp  time
1  slow  hot  long
2  fast  hot short
3  slow cold short
4  fast cold  long
class=design, type= FrF2
```

```
> #
```

In fact, FrF2 has dozens of optional arguments, and we will only scratch the surface of what it can do.

```
> design.info(des1)
```

You can get information about a design using the design.info function. There's probably more information than you really want.

```
$type
[1] "FrF2"

$nruns
[1] 16

$nfactors
[1] 5

$factor.names
$factor.names$A
[1] -1  1

$factor.names$B
[1] -1  1

$factor.names$C
[1] -1  1

$factor.names$D
[1] -1  1

$factor.names$E
[1] -1  1


$catlg.name
[1] "catlg"

$catlg.entry
Design:  5-1.1
   16  runs,  5  factors,
   Resolution  V
   Generating columns:  15
   WLP (3plus):  0 0 1 0 0 ,  10  clear 2fis

$aliased
$aliased$legend
[1] "A=A" "B=B" "C=C" "D=D" "E=E"

$aliased[[2]]
[1] "no aliasing among main effects and 2fis"


$FrF2.version
[1] "1.1-5"

$replications
[1] 1
```

```
$repeat.only
[1] FALSE

$randomize
[1] TRUE

$seed
NULL

$creator
FrF2(16, 5)
```

> **design.info(des1)$catlg.entry**

Most of these designs are going to come from a catalog of designs (like the tables in the back of the book). This command just strips out the catalog information. Some interesting bits include the resolution (here V), and the generating columns.

A 2^{n-k} design is set up as complete factorials in the first $n-k$ factors. Each of the k additional factors is aliased to an interaction of the first $n-k$ factors. The "Generating columns" information tells us which columns are the generator; here it is just column 15. These columns are the +1/–1 columns for the various interactions, and they are numbered in standard order beginning with A. Number 15 is the ABCD interaction, so column E (the fifth factor) is generated by ABCD, yielding the overall generator I=ABCDE, giving us resolution V.

```
Design:  5-1.1
   16  runs,  5  factors,
   Resolution  V
   Generating columns:  15
   WLP (3plus):  0 0 1 0 0 ,  10  clear 2fis
```

> **des2 <- FrF2(8,5);des2**

Now let's look at something with more fractioning. Here we have five factors in 8 runs, so this is a quarter fraction that will have two generators.

```
   A  B  C  D  E
1 -1 -1  1  1 -1
2  1  1 -1  1 -1
3 -1  1  1 -1 -1
4 -1  1 -1 -1  1
5  1 -1 -1 -1 -1
6  1  1  1  1  1
7 -1 -1 -1  1  1
8  1 -1  1 -1  1
class=design, type= FrF2
```

```
> FrF2(8,5,randomize=FALSE)
```

FrF2 works by taking a complete factorial in the first $n - k$ variables and then adding the additional k variables on using the aliasing. However, it randomizes the order of the runs, so it is often challenging to see the base factorial. If you tell it not to randomize, then you can clearly see the base factorial in standard order.

```
   A  B  C  D  E
1 -1 -1 -1  1  1
2  1 -1 -1 -1 -1
3 -1  1 -1 -1  1
4  1  1 -1  1 -1
5 -1 -1  1  1 -1
6  1 -1  1 -1  1
7 -1  1  1 -1 -1
8  1  1  1  1  1
class=design, type= FrF2
```

```
> design.info(des2)$catlg.entry
```

The generating columns are 3 (AB) and 5 (AC), so the aliasing is based on I = ABD = ACE = BCDE. Note the WLP output. It tells us how many three letter, four letter, and so on words are aliased with I, here 2 three-way interactions and one four-way, as we just saw. It also tells us how many two factor interactions are "clear," that is, not aliased to main effects or other two factor interactions. In this case, all two factor interactions are aliased to a main effect, so none of them were clear. Just above in des1, all two factor interactions were aliased with three factor interactions, so all (10) of them were clear.

```
Design:  5-2.1
   8  runs,  5  factors,
   Resolution  III
   Generating columns:  3 5
   WLP (3plus):  2 1 0 0 0 ,  0  clear 2fis
```

```
> FrF2(8,generators=c("AC","BC"),randomize=FALSE)
```

Here we ask for a design with generators ACD and BCE, so ABDE is also aliased.

```
   A  B  C  D  E
1 -1 -1 -1  1  1
2  1 -1 -1 -1  1
3 -1  1 -1  1 -1
4  1  1 -1 -1 -1
5 -1 -1  1 -1 -1
6  1 -1  1  1 -1
7 -1  1  1 -1  1
8  1  1  1  1  1
class=design, type= FrF2.generators
```

```
> FrF2(8,generators=c("AC","-BC"),randomize=FALSE)
```

We can also change signs to get alternate fractions. Here we flip the signs for E.

```
   A  B  C  D  E
1 -1 -1 -1  1 -1
2  1 -1 -1 -1 -1
3 -1  1 -1  1  1
4  1  1 -1 -1  1
5 -1 -1  1 -1  1
6  1 -1  1  1  1
7 -1  1  1 -1 -1
8  1  1  1  1 -1
class=design, type= FrF2.generators
```

```
> FrF2(8,generators=c("AC","BC"),factor.nam=c("A","D","C","B","E"),randomize=FALSE)
```

FrF2 always assumes that we have a complete factorial in the first three factors. That means that we could never use ABC as one of the generators. We can get around that by relabeling the factors. In the generator, the AC here really means interaction of first and third factor aliased to the fourth factor; at this stage, FrF2 doesn't really care how you eventually label the columns. The generators make sure that the fourth column is aliased to the interaction of the first and the third. If the factors are renamed as above, the first, third, and fourth columns are A, B, and C, so we get ABC as one of the aliases of I. Note below that we do not have a full factorial in ABC below; ABC is always positive.

```
   A  D  C  B  E
1 -1 -1 -1  1  1
2  1 -1 -1 -1  1
3 -1  1 -1  1 -1
4  1  1 -1 -1 -1
5 -1 -1  1 -1 -1
6  1 -1  1  1 -1
7 -1  1  1 -1  1
8  1  1  1  1  1
class=design, type= FrF2.generators
```

```
> FrF2(8,generators=c(5,6),factor.nam=c("A","D","C","B","E"),randomize=FALSE)
```

We get the same thing using 5 and 6 instead of AC and BC. Recall standard order: A, B, AB, C, AC, BC, ABC, . . .; AC is fifth in that order and BC is sixth. But don't think about actual factor names, think of it as first column, second column, interaction of first and second, third column, and so on. Then FrF2 adds on the names later.

```
   A  D  C  B  E
1 -1 -1 -1  1  1
2  1 -1 -1 -1  1
3 -1  1 -1  1 -1
4  1  1 -1 -1 -1
5 -1 -1  1 -1 -1
6  1 -1  1  1 -1
7 -1  1  1 -1  1
8  1  1  1  1  1
class=design, type= FrF2.generators
```

```
> des3 <- FrF2(16,8,randomize=FALSE);des3
```

Here is a more interesting design, a 2^{8-4}, which is 8 factors in 16 runs for a 1/16 fraction.

```
    A  B  C  D  E  F  G  H
1  -1 -1 -1 -1 -1 -1 -1 -1
2   1 -1 -1 -1  1  1  1 -1
3  -1  1 -1 -1  1  1 -1  1
4   1  1 -1 -1 -1 -1  1  1
5  -1 -1  1 -1  1 -1  1  1
6   1 -1  1 -1 -1  1 -1  1
7  -1  1  1 -1 -1  1  1 -1
8   1  1  1 -1  1 -1 -1 -1
9  -1 -1 -1  1 -1  1  1  1
10  1 -1 -1  1  1 -1 -1  1
11 -1  1 -1  1  1 -1  1 -1
12  1  1 -1  1 -1  1 -1 -1
13 -1 -1  1  1  1  1 -1 -1
14  1 -1  1  1 -1 -1  1 -1
15 -1  1  1  1 -1 -1 -1  1
16  1  1  1  1  1  1  1  1
class=design, type= FrF2
```

```
> design.info(des3)$catlg.entry
```

The generating columns are all three factor interactions [7=ABC, 11=ABD, 13=ACD, 14=BCD], which gives us four factor interactions (and the eight-way interaction) as the aliases of I. Note the resolution IV.

A counts as 1; B counts as 2; C counts as 4; D counts as 8; and so on. Thus ABD is $1+2+8=11$.

```
Design:  8-4.1
   16  runs,  8  factors,
   Resolution  IV
   Generating columns:  7 11 13 14
   WLP (3plus):  0 14 0 0 0 ,  0  clear 2fis
```

```
> des2 <- within(des2,y <- rnorm(8));fit2 <- lm(y~A*B*C*D*E,data=des2)
```

There are functions for determining the alias structure, but they all use the output of lm(). To use them, we'll make a dummy response for our design 2 with 8 runs and then fit the model with all main effects and interactions. Clearly, we cannot actually fit all 31 main effects and interactions with only 8 data points.

```
> anova(fit2)
```

We've tried to fit a mean and 31 effects to 8 data points, so only a few things are going to be estimable. The way lm() does things is that it does main effects, then 2fis, etc. It can't fit AB because it's aliased with D, it can't fit AC because it's aliased with E; it can fit BC, and eventually CD.

```
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
A          1 0.12543 0.12543
B          1 0.63095 0.63095
C          1 0.68115 0.68115
D          1 0.48364 0.48364
```

```
E          1 2.29482 2.29482
B:C        1 0.34818 0.34818
C:D        1 0.79358 0.79358
Residuals  0 0.00000
Warning message:
In anova.lm(fit2) :
  ANOVA F-tests on an essentially perfect fit are unreliable
```

```
> fit2

Call:
lm.default(formula = y ~ A * B * C * D * E, data = des2)

Coefficients:
   (Intercept)              A1              B1              C1
        0.4561          0.1252          0.2808         -0.2918
            D1              E1           A1:B1           A1:C1
       -0.2459         -0.5356              NA              NA
         B1:C1           A1:D1           B1:D1           C1:D1
        0.2086              NA              NA         -0.3150
         A1:E1           B1:E1           C1:E1           D1:E1
            NA              NA              NA              NA
      A1:B1:C1        A1:B1:D1        A1:C1:D1        B1:C1:D1
            NA              NA              NA              NA
      A1:B1:E1        A1:C1:E1        B1:C1:E1        A1:D1:E1
            NA              NA              NA              NA
      B1:D1:E1        C1:D1:E1     A1:B1:C1:D1     A1:B1:C1:E1
            NA              NA              NA              NA
   A1:B1:D1:E1     A1:C1:D1:E1     B1:C1:D1:E1  A1:B1:C1:D1:E1
            NA              NA              NA              NA
```

```
> aliases(fit2)
```

The aliases() function gives us the full table of aliases, except that it does not give the aliases of I. You can find those easily from any row of the output by multiplying by the first element and reducing exponents modulo 2. This gives us I = A:C:E = B:C:D:E = A:B:D.

```
A = C:E = A:B:C:D:E = B:D
B = C:D:E = A:B:C:E = A:D
C = A:E = B:D:E = A:B:C:D
D = B:C:E = A:C:D:E = A:B
E = B:C:D = A:B:D:E = A:C
B:C = D:E = A:C:D = A:B:E
C:D = B:E = A:B:C = A:D:E
```

```
> alias(fit2)
```

The alias function presents the same information in a different format. There is a column for each fitted term and a row for each aliased term. We then have 1's wherever an aliased term is aliased to a fitted term.

```
Model :
y ~ A * B * C * D * E

Complete :
               (Intercept) A1 B1 C1 D1 E1 B1:C1 C1:D1
A1:E1          0           0  0  1  0  0  0      0
B1:E1          0           0  0  0  0  0  0      1
C1:E1          0           1  0  0  0  0  0      0
D1:E1          0           0  0  0  0  0  1      0
A1:B1:C1       0           0  0  0  0  0  0      1
A1:B1:D1       1           0  0  0  0  0  0      0
A1:C1:D1       0           0  0  0  0  0  1      0
B1:C1:D1       0           0  0  0  0  1  0      0
A1:B1:E1       0           0  0  0  0  0  1      0
A1:C1:E1       1           0  0  0  0  0  0      0
B1:C1:E1       0           0  0  0  1  0  0      0
A1:D1:E1       0           0  0  0  0  0  0      1
B1:D1:E1       0           0  0  1  0  0  0      0
C1:D1:E1       0           0  1  0  0  0  0      0
A1:B1:C1:D1    0           0  0  1  0  0  0      0
A1:B1:C1:E1    0           0  1  0  0  0  0      0
A1:B1:D1:E1    0           0  0  0  0  1  0      0
A1:C1:D1:E1    0           0  0  0  1  0  0      0
B1:C1:D1:E1    1           0  0  0  0  0  0      0
A1:B1:C1:D1:E1 0           1  0  0  0  0  0      0
A1:B1          0           0  0  0  1  0  0      0
A1:C1          0           0  0  0  0  1  0      0
A1:D1          0           0  1  0  0  0  0      0
B1:D1          0           1  0  0  0  0  0      0
```

```
> des3 <- within(des3,y<-rnorm(16));fit3 <- lm(y~A*B*C*D*E*F*G*H,data=des3)
> aliases(fit3)
```

We can get aliases for bigger designs, but the output can be enormous. (Or, perhaps very tiny if you want to see it all.)

```
A = B:C:E = B:D:F = C:D:G = E:F:G = D:E:H = C:F:H = B:G:H = A:C:D:E:F = A:B:D:E:G = A:B:C:F:G = A:B:C:D:H = A:B:E:F:H = A:C:E:G:H = A:D:F:G:H = B:C:D:E:F:G:H
B = A:C:E = A:D:F = D:E:G = C:F:G = C:D:H = E:F:H = A:G:H = B:C:D:E:F = A:B:C:D:G = A:B:E:F:G = A:B:D:E:H = A:B:C:F:H = B:C:E:G:H = B:D:F:G:H = A:C:D:E:F:G:H
C = A:B:E = D:E:F = A:D:G = B:F:G = B:D:H = A:F:H = E:G:H = A:B:C:D:F = B:C:D:E:G = A:C:E:F:G = A:C:D:E:H = B:C:E:F:H = A:B:C:G:H = C:D:F:G:H = A:B:D:E:F:G:H
D = A:B:F = C:E:F = A:C:G = B:E:G = B:C:H = A:E:H = F:G:H = A:B:C:D:E = B:C:D:F:G = A:D:E:F:G = A:C:D:F:H = B:D:E:F:H = A:B:D:G:H = C:D:E:G:H = A:B:C:E:F:G:H
E = A:B:C = C:D:F = B:D:G = A:F:G = A:D:H = B:F:H = C:G:H = A:B:D:E:F = A:C:D:E:G = B:C:E:F:G = B:C:D:E:H = A:C:E:F:H = A:B:E:G:H = D:E:F:G:H = A:B:C:D:F:G:H
F = A:B:D = C:D:E = B:C:G = A:E:G = A:C:H = B:E:H = D:G:H = A:B:C:E:F = A:C:D:F:G = B:D:E:F:G = B:C:D:F:H = A:D:E:F:H = A:B:F:G:H = C:E:F:G:H = A:B:C:D:E:G:H
G = A:C:D = B:D:E = B:C:F = A:E:F = A:B:H = C:E:H = D:F:H = A:B:C:E:G = A:B:D:F:G = C:D:E:F:G = B:C:D:G:H = A:D:E:G:H = A:C:F:G:H = B:E:F:G:H = A:B:C:D:E:F:H
H = B:C:D = A:D:E = A:C:F = B:E:F = A:B:G = C:E:G = D:F:G = A:B:C:E:H = A:B:D:F:H = C:D:E:F:H = A:C:D:G:H = B:D:E:G:H = B:C:F:G:H = A:E:F:G:H = A:B:C:D:E:F:G
A:B = D:F = G:H = B:C:D:G = A:D:E:G = A:C:F:G = B:E:F:G = A:C:D:H = B:D:E:H = B:C:F:H = A:E:F:H = A:B:C:D:E:F = A:B:C:E:G:H = A:B:D:F:G:H = C:D:E:F:G:H = C:E
A:C = D:G = F:H = B:C:D:F = A:D:E:F = A:B:F:G = C:E:F:G = A:B:D:H = C:D:E:H = B:C:G:H = A:E:G:H = A:B:C:D:E:G = A:B:C:E:F:H = A:C:D:F:G:H = B:D:E:F:G:H = B:E
B:C = F:G = D:H = A:C:D:F = B:D:E:F = A:B:D:G = C:D:E:G = A:B:F:H = C:E:F:H = A:C:G:H = B:E:G:H = A:B:C:E:F:G = A:B:C:D:E:H = B:C:D:F:G:H = A:D:E:F:G:H = A:E
A:D = B:F = C:G = E:H = B:C:D:E = A:C:E:F = A:B:E:G = D:E:F:G = A:B:C:H = C:D:F:H = B:D:G:H = A:F:G:H = A:B:C:D:F:G = A:B:D:E:F:H = A:C:D:E:G:H = B:C:E:F:G:H
B:D = A:F = E:G = C:H = A:C:D:E = B:C:E:F = A:B:C:G = C:D:F:G = A:B:E:H = D:E:F:H = A:D:G:H = B:F:G:H = A:B:D:E:F:G = A:B:C:D:F:H = B:C:D:E:G:H = A:C:E:F:G:H
C:D = E:F = A:G = B:H = A:B:D:E = A:B:C:F = B:C:E:G = B:D:F:G = A:C:E:H = A:D:F:H = D:E:G:H = C:F:G:H = A:C:D:E:F:G = B:C:D:E:F:H = A:B:C:D:G:H = A:B:E:F:G:H
D:E = C:F = B:G = A:H = A:B:C:D = A:B:E:F = A:C:E:G = A:D:F:G = B:C:E:H = B:D:F:H = C:D:G:H = E:F:G:H = B:C:D:E:F:G = A:C:D:E:F:H = A:B:D:E:G:H = A:B:C:F:G:H
```

```
> des2 <- within(des2,{y2 <- y;y2[1]<-NA})
```

Missing data change the aliasing structure, and usually make it much more complicated. Here we will just make a second response that has one missing value.

```
> fit2b <- lm(y2~A*B*C*D*E,data=des2)
> alias(fit2b)
```

Note that there are only six columns for estimable effects, instead of the seven we had before. Note also that four of the effects (CD, BE, ABC, and ADE) are now aliased to linear combinations of all the other effects. Very messy.

```
Model :
y ~ A * B * C * D * E

Complete :
               (Intercept) A1 B1 C1 D1 E1 B1:C1
A1:D1           0           0  1  0  0  0  0
B1:D1           0           1  0  0  0  0  0
C1:D1          -1          -1  1  1  1  1 -1
A1:E1           0           0  0  1  0  0  0
B1:E1          -1          -1  1  1  1  1 -1
C1:E1           0           1  0  0  0  0  0
D1:E1           0           0  0  0  0  0  1
A1:B1:C1       -1          -1  1  1  1  1 -1
A1:B1:D1        1           0  0  0  0  0  0
A1:C1:D1        0           0  0  0  0  0  1
B1:C1:D1        0           0  0  0  0  1  0
A1:B1:E1        0           0  0  0  0  0  1
A1:C1:E1        1           0  0  0  0  0  0
B1:C1:E1        0           0  0  0  1  0  0
A1:D1:E1       -1          -1  1  1  1  1 -1
B1:D1:E1        0           0  0  1  0  0  0
C1:D1:E1        0           0  1  0  0  0  0
A1:B1:C1:D1     0           0  0  1  0  0  0
A1:B1:C1:E1     0           0  1  0  0  0  0
A1:B1:D1:E1     0           0  0  0  0  1  0
A1:C1:D1:E1     0           0  0  0  1  0  0
B1:C1:D1:E1     1           0  0  0  0  0  0
A1:B1:C1:D1:E1  0           1  0  0  0  0  0
A1:B1           0           0  0  0  1  0  0
A1:C1           0           0  0  0  0  1  0
```

```
> #
```

Speaking of messy, before moving on to data, let's look at another class of fractional factorial designs called Plackett-Burman designs. They are resolution III designs in k=N-1 factors in N data points when N is a multiple of 4. In this example, we look at 11 factors in 12 runs.

```
> A <- factor(c(2,2,1,2,2,2,1,1,1,2,1,1))
> B <- factor(c(1,2,2,1,2,2,2,1,1,1,2,1))
> C <- factor(c(2,1,2,2,1,2,2,2,1,1,1,1))
> D <- factor(c(1,2,1,2,2,1,2,2,2,1,1,1))
> E <- factor(c(1,1,2,1,2,2,1,2,2,2,1,1))
> F <- factor(c(1,1,1,2,1,2,2,1,2,2,2,1))
> G <- factor(c(2,1,1,1,2,1,2,2,1,2,2,1))
> H <- factor(c(2,2,1,1,1,2,1,2,2,1,2,1))
```

```
> I <- factor(c(2,2,2,1,1,1,2,1,2,2,1,1))
> J <- factor(c(1,2,2,2,1,1,1,2,1,2,2,1))
> K <- factor(c(2,1,2,2,2,1,1,1,2,1,2,1))
> pb <- data.frame(A,B,C,D,E,F,G,H,I,J,K,y=rnorm(12))
> outpb <- lm(y~(A+B+C+D+E+F+G+H+I+J+K)^2,data=pb)
```

In principle we could include all 4047 main effects and interactions, but I just went up to 2 factor interactions, as they are sufficient to illustrate the point.

```
> alias(outpb)
```

Here we look at aliasing. *Every second order interaction is aliased to all main effects not occurring in the interaction.* Thus you had better be really, really sure about not having any sizable interactions when you use a PB design, because a single non-ignorable interaction screws up all of your main effects estimates, not just one.

```
Model :
y ~ (A + B + C + D + E + F + G + H + I + J + K)^2

Complete :
      (Intercept) A1   B1   C1   D1   E1   F1   G1   H1   I1   J1   K1
A1:B1           0  0    0  1/3 -1/3 -1/3  1/3  1/3 -1/3  1/3  1/3  1/3
A1:C1           0  0  1/3    0  1/3  1/3 -1/3  1/3 -1/3  1/3  1/3 -1/3
A1:D1           0  0 -1/3  1/3    0  1/3  1/3  1/3  1/3  1/3 -1/3 -1/3
A1:E1           0  0 -1/3  1/3  1/3    0 -1/3 -1/3  1/3  1/3  1/3  1/3
A1:F1           0  0  1/3 -1/3  1/3 -1/3    0  1/3  1/3  1/3 -1/3  1/3
A1:G1           0  0  1/3  1/3  1/3 -1/3  1/3    0  1/3 -1/3  1/3 -1/3
A1:H1           0  0 -1/3 -1/3  1/3  1/3  1/3  1/3    0 -1/3  1/3  1/3
A1:I1           0  0  1/3  1/3  1/3  1/3  1/3 -1/3 -1/3    0 -1/3  1/3
A1:J1           0  0  1/3  1/3 -1/3  1/3 -1/3  1/3  1/3 -1/3    0  1/3
A1:K1           0  0  1/3 -1/3 -1/3  1/3  1/3 -1/3  1/3  1/3  1/3    0
B1:C1           0  1/3    0    0  1/3 -1/3 -1/3  1/3  1/3 -1/3  1/3  1/3
B1:D1           0 -1/3    0  1/3    0  1/3  1/3 -1/3  1/3 -1/3  1/3  1/3
B1:E1           0 -1/3    0 -1/3  1/3    0  1/3  1/3  1/3  1/3  1/3 -1/3
B1:F1           0  1/3    0 -1/3  1/3  1/3    0 -1/3 -1/3  1/3  1/3  1/3
B1:G1           0  1/3    0  1/3 -1/3  1/3 -1/3    0  1/3  1/3  1/3 -1/3
B1:H1           0 -1/3    0  1/3  1/3  1/3 -1/3  1/3    0  1/3 -1/3  1/3
B1:I1           0  1/3    0 -1/3 -1/3  1/3  1/3  1/3  1/3    0 -1/3  1/3
B1:J1           0  1/3    0  1/3  1/3  1/3  1/3  1/3 -1/3 -1/3    0 -1/3
B1:K1           0  1/3    0  1/3  1/3 -1/3  1/3 -1/3  1/3  1/3 -1/3    0
C1:D1           0  1/3  1/3    0    0  1/3 -1/3 -1/3  1/3  1/3 -1/3  1/3
C1:E1           0  1/3 -1/3    0  1/3    0  1/3  1/3 -1/3  1/3 -1/3  1/3
C1:F1           0 -1/3 -1/3    0 -1/3  1/3    0  1/3  1/3  1/3  1/3  1/3
C1:G1           0  1/3  1/3    0 -1/3  1/3  1/3    0 -1/3 -1/3  1/3  1/3
C1:H1           0 -1/3  1/3    0  1/3 -1/3  1/3 -1/3    0  1/3  1/3  1/3
C1:I1           0  1/3 -1/3    0  1/3  1/3  1/3 -1/3  1/3    0  1/3 -1/3
C1:J1           0  1/3  1/3    0 -1/3 -1/3  1/3  1/3  1/3  1/3    0 -1/3
C1:K1           0 -1/3  1/3    0  1/3  1/3  1/3  1/3  1/3 -1/3 -1/3    0
D1:E1           0  1/3  1/3  1/3    0    0  1/3 -1/3 -1/3  1/3  1/3 -1/3
D1:F1           0  1/3  1/3 -1/3    0  1/3    0  1/3  1/3 -1/3  1/3 -1/3
D1:G1           0  1/3 -1/3 -1/3    0 -1/3  1/3    0  1/3  1/3  1/3  1/3
D1:H1           0  1/3  1/3  1/3    0 -1/3  1/3  1/3    0 -1/3 -1/3  1/3
```

```
> y<-c(40,20,17,12,31,19,22,20,36,25,34,11,37,21,29,19)
```
Data (in standard order) from a 2^{5-1} from Garcia-Diaz and Phillips.

```
> A <- factor(rep(1:2,each=1,length=16))
> B <- factor(rep(1:2,each=2,length=16))
> C <- factor(rep(1:2,each=4,length=16))
> D <- factor(rep(1:2,each=8,length=16))
> E <- factor(c(2,1,1,2,1,2,2,1,1,2,2,1,2,1,1,2))
```
I enter E manually to reflect the pattern implied by the aliasing.

```
> fitgd <- lm(y~A*B*C*D*E)
```
Fit the full model.

```
> aliases(fitgd)
```
So we have I=ABCDE, which is what we would expect.

```
 A = B:C:D:E
 B = A:C:D:E
 C = A:B:D:E
 D = A:B:C:E
 E = A:B:C:D
 A:B = C:D:E
 A:C = B:D:E
 B:C = A:D:E
 A:D = B:C:E
 B:D = A:C:E
 C:D = A:B:E
 A:E = B:C:D
 B:E = A:C:D
 C:E = A:B:D
 D:E = A:B:C
```

```
> anova(fitgd)
```
We can fit the main effects and all two-factor interactions.

```
          Df Sum Sq Mean Sq F value Pr(>F)
A          1 612.56  612.56
B          1 264.06  264.06
C          1   0.56    0.56
D          1  60.06   60.06
E          1  33.06   33.06
A:B        1  22.56   22.56
A:C        1  22.56   22.56
B:C        1  52.56   52.56
A:D        1  27.56   27.56
B:D        1  10.56   10.56
C:D        1   0.56    0.56
A:E        1  18.06   18.06
B:E        1   0.56    0.56
C:E        1  60.06   60.06
D:E        1  10.56   10.56
Residuals  0   0.00
Warning message:
In anova.lm(fitgd) :
  ANOVA F-tests on an essentially perfect fit are unreliable
```

```
> anova(lm(y~A*B*C*D))
```

We can fit the full A*B*C*D model, because we have a full factorial in A, B, C, D (actually any four factors in this design). Note that we get the same SS as in the previous anova, the labels just change but aliased terms have the same SS.

```
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
A          1 612.56  612.56
B          1 264.06  264.06
C          1   0.56    0.56
D          1  60.06   60.06
A:B        1  22.56   22.56
A:C        1  22.56   22.56
B:C        1  52.56   52.56
A:D        1  27.56   27.56
B:D        1  10.56   10.56
C:D        1   0.56    0.56
A:B:C      1  10.56   10.56
A:B:D      1  60.06   60.06
A:C:D      1   0.56    0.56
B:C:D      1  18.06   18.06
A:B:C:D    1  33.06   33.06
Residuals  0   0.00
Warning message:
In anova.lm(lm(y ~ A * B * C * D)) :
  ANOVA F-tests on an essentially perfect fit are unreliable
```

```
> anova(lm(y~B*C*D*E))
```

Ditto for B*C*D*E; same SS, just in different orders and different labels based on aliasing.

```
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
B          1 264.06  264.06
C          1   0.56    0.56
D          1  60.06   60.06
E          1  33.06   33.06
B:C        1  52.56   52.56
B:D        1  10.56   10.56
C:D        1   0.56    0.56
B:E        1   0.56    0.56
C:E        1  60.06   60.06
D:E        1  10.56   10.56
B:C:D      1  18.06   18.06
B:C:E      1  27.56   27.56
B:D:E      1  22.56   22.56
C:D:E      1  22.56   22.56
B:C:D:E    1 612.56  612.56
Residuals  0   0.00
Warning message:
In anova.lm(lm(y ~ B * C * D * E)) :
  ANOVA F-tests on an essentially perfect fit are unreliable
```

```
> anova(lm(y~A+B+C+D+E))
```

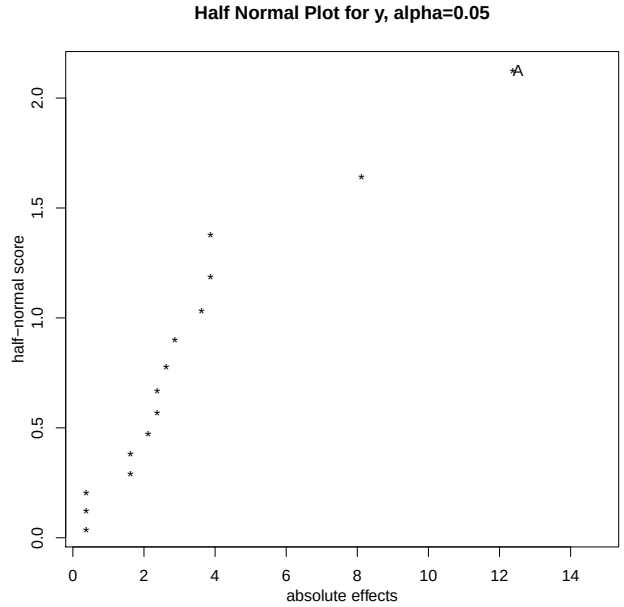
If we are going to try to analyze this by anova, we'd likely just look at main effects. It looks like A and B are significant. Keep in mind, however, that this error isn't really pure error, it is a pooling of interaction terms.

```
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value    Pr(>F)
A          1 612.56  612.56 27.1496 0.0003952 ***
B          1 264.06  264.06 11.7036 0.0065359 **
C          1   0.56    0.56  0.0249 0.8776826
D          1  60.06   60.06  2.6620 0.1338198
E          1  33.06   33.06  1.4654 0.2539113
Residuals 10 225.62   22.56
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> DanielPlot(fitgd,half=TRUE)
```

I prefer to look at a Daniel plot or halfnormal plot of the effects. Only A is selected by the Lenth criterion, but I have to say, B looks like an outlier to me as well.




```
> y<-c(142,106,88,109,113,162,200,79,101,108,146,72,200,83,115,118)
```

These data are again a 2^{5-1} from Davies, but run in two blocks of size 8 with ABD=–CE confounded with blocks. Factors A, B, C, and D are the same as before, due to standard order.

```
> E<-factor(c(1,2,2,1,2,1,1,2,2,1,1,2,1,2,2,1))
```

The generator here is E=–ABCDE.

```
> bl<-factor(c(1,2,2,1,1,2,2,1,2,1,1,2,2,1,1,2))
```

ABD=–CE as block.

```
> fitd <- lm(y~bl+A+B+C+D+E);anova(fitd)
```

A preliminary fit with blocks and main effects finds some significant main effects.

```
Analysis of Variance Table

Response: y
          Df  Sum Sq Mean Sq F value    Pr(>F)
bl         1  1444.0  1444.0  5.2840 0.0470998 *
A          1  4489.0  4489.0 16.4265 0.0028721 **
B          1   484.0   484.0  1.7711 0.2159763
C          1  2450.2  2450.2  8.9662 0.0150940 *
D          1   196.0   196.0  0.7172 0.4190069
E          1 11449.0 11449.0 41.8951 0.0001151 ***
Residuals  9  2459.5   273.3
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> fitd2 <- lm(y~bl+A*C*E);anova(fitd2)
```

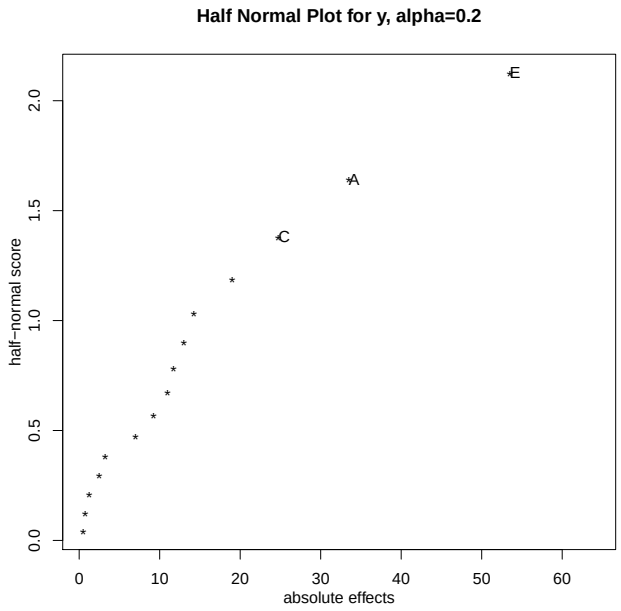
Project down onto A, C, and E. It still only looks like main effects. Note that CE is not fit, because CE is confounded with blocks.

```
Analysis of Variance Table

Response: y
          Df  Sum Sq Mean Sq F value    Pr(>F)
bl         1  1444.0  1444.0  7.0055  0.029402 *
A          1  4489.0  4489.0 21.7780  0.001609 **
C          1  2450.3  2450.3 11.8872  0.008723 **
E          1 11449.0 11449.0 55.5440 7.245e-05 ***
A:C        1   676.0   676.0  3.2796  0.107733
A:E        1   812.3   812.3  3.9406  0.082397 .
A:C:E      1     2.3     2.3  0.0109  0.919362
Residuals  8  1649.0   206.1
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
> fitdall <- lm(y~A*B*C*D*E)
> DanielPlot(fitdall,half=TRUE,alpha=.2)
```

You need to make alpha pretty big before Lenth marks all three main effects as significant. I'm inclined to say that E and A are outliers and C is borderline. The next biggest one is CE, which is block and so not really something to test.


Half Normal Plot for y, alpha=0.2
half−normal score
absolute effects
0.0
0.5
1.0
1.5
2.0
0
10
20
30
40
50
60
E
A
C