STATISTICAL MODELLING

Appendix C Analysis of designed experiments in R

C.1. Entering the results of an experiment into a data.frame

To decide how to enter the results into a data frame, the first question is do you already have a data frame with the factors entered. If yes, then use a) below. If no, then use b) if all the factors are in standard order and c) if they are not.

a) Adding the response variable to a randomized layout

In some cases you will have generated a randomized layout, using `fac.layout` as described in Appendix B, *Randomized layouts and sample size computations in R*, so that the factors specifying the layout are stored in a `data.frame`. If this is the case, then it will simply be a matter of adding the values of the response variable(s) in random order. This might be done as follows:

```
y <- c( … )
RCDB.dat <- data.frame(RCDB.lay, y)
```

That is, the `c` function is used to create a vector with the response variable and then it and the `data.frame` containing the layout are combined and assigned to a new `data.frame`.

b) Creating a data.frame from scratch with the factors in standard order

If you have to create a `data.frame` from scratch and the data has been provided in standard order, then it will be a matter of generating the factors in standard order and adding the data. There are a number of methods for doing this and they involve using the following functions for the purpose indicated (for more details see Appendix B or usew the `help` function:

`c`:	create the response variable;
`factor`:	create `factors`;
`rep`:	generate the values for a `factor` when they follow a pattern;
`fac.gen`:	generate the values for a set of `factors`;
`data.frame`:	to create `data.frames` by combining `factors`, `vectors` and `data.frames`.

In addition the construction `data.frame$name <-` … will be used to assign an object to a column, named `name`, in a `data.frame`.

Some methods for creating the `data.frame` are now outlined, using as an example randomized complete block design. It is supposed that there are *b* blocks each with *t* treatments.

1. **Assign the factors and data to a `data.frame`**. For example,

```
RCDB.dat <- data.frame(
                Block = factor(rep(1:b, each=t)),
                Unit = factor(rep(1:t, times=b)),
                 Treatment = factor(rep(1:t, times=b)))
RCDB.dat$y <- c( … )
```

You may also want to include labels for the levels of the factors (e.g. I, II, III, … etc.) using the `labels` argument.

2. **Generate the unrandomized of factors in standard order, in a `data.frame`, and then add rest** (other factors and the response variable) to this data frame. For example, in a randomized complete block design the unrandomized factors Blocks and Units will uniquely index the units of the experiment. These two factors can be generated using the fac.gen function as follows:

```
RCBD.dat <- fac.gen(list(Blocks = b, Units = t))
```

Now the Treatments factor and response variable y need to be added. There are two ways to do this.

i. Direct addition

```
RCDB.dat$Treatment <- factor(rep(1:t, times=b))
RCDB.dat$y <- c( … )
```

ii. Indirect addition

```
Treatment <- factor(rep(1:t, times=b))
y <- c( … )
RCDB.dat <- data.frame(RCDB.dat, Treatment, y)
```

You may also want to include labels for the levels of the factors (e.g. I, II, III, … etc.) using the `labels` argument.

3. **Generate the treatment factors in standard order, in a `data.frame`, and then add rest** (unrandomized factors and the response variable) to this data frame. This is most likely for a factorial experiment, laid out as a completely randomized design. Such experiments have only a single unrandomized factor, say Runs. For example, suppose the four treatment factors are A, B, C and D each with 2 levels so that the number of treatments is $t = 2^4$. In this case, we will use the labels '-' and '+' for the levels. Also, suppose that each treatment is replicated r times in a completely randomized design consisting of $8r$ runs. In general, I follow the convention that the unrandomized factors precede the randomized factors in the data frame. The following commands do this, and add a response variable too.

```
n <- 8 * r
mp <- c("-", "+")
Fac4.Rep.dat <- fac.gen(generate
                 = list(A = mp, B = mp, C = mp, D = mp),
                each = r, order="yates")
Fac4.Rep.dat <- data.frame(Runs = factor(1:n),
                           Fac4.Rep.dat)
Fac4.Rep.dat$y <- c( … )
```

4. **Generate two sets of factors in two separate `data.frames`** and concatenate these before adding the response variable. This will be particularly relevant for factorial experiments laid out using other than a completely randomized design. For example, in a randomized complete block design the unrandomized factors Blocks and Units will uniquely index the units of the experiment. These two factors can be generated using the fac.gen function as follows:

```
RCBD.dat <- fac.gen(list(Blocks = b, Units = t))
RCBD.trt <- fac.gen(list(C = c, D = d), times = b)
y <- c( … )
RCDB.dat <- data.frame(RCDB.dat, RCDB.trt, y)
```

You may also want to include labels for the levels of the factors (e.g. I, II, III, … etc.) using the `labels` argument.

c) Creating a data.frame from scratch with the data recorded against the randomized layout

If you have to create a `data.frame` from scratch and the data has been recorded against the randomized layout, then you can use either of the first two methods described in b). However, the *Treatment* factor will not be able to be generated using the `rep` function. Instead, the levels in the randomized order will be listed inside a `c` function, in the same way as *y*.

C.2. The elements of the analysis of experiments

Generally the analysis begins with an initial graphical exploration of the data — boxplots for experiments with a single treatment factor and interaction plots for factorial experiments. Then the `aov` function is used to analyse the data from the experiment. The analysis is specified using a *model formula* of the form:

Response variable ~ *explanatory variables* (and *operators*)

There is a subtlety in the type of the explanatory variables that arises in connection with the analysis of designed experiments. If the explanatory variable is a numeric, such as a numeric vector, then R fits just one coefficient for it. So for a single explanatory variable, a straight-line relationship between the response and explanatory variables is fitted. On the other hand, if the explanatory variable is categorical, such as a factor, a coefficient is fit for each level of the variable. Being stored in a factor object signals to R that it should use indicator variables, or their equivalent, instead of values for the variable itself in estimating the parameters. Most often explanatory variables will be factors. Also different forms of the analysis are obtained depending on whether the `Error` function is used as part of the explanatory variable specification. Generally, the form with the `Error` function is preferred. When you use the `Error` function you must also include outside the `Error` function those fixed terms from inside the `Error` function. The `aov` function produces an `aovlist` object when the `Error` function is used — it is a `list` of `aov` objects, one for each `Error` term in the analysis. Without the `Error` function a single `aov` object is produced.

Next diagnostic checking based on the residuals is performed — residual-versus-fitted-values and residual-versus-factors plots, normal probability plot of the residuals and, for all but the CRD, Tukey's one-degree-of-freedom-for-nonadditivity. The functions `resid.errors` and `fitted.errors` must be used to extract the residuals and fitted values when the `Error` function is used in a call to `aov`. Also the function `tukey.1df` performs the nonadditivity test — you must specify the `error.term` when using `tukey.1df` following an `aov` function in which and `Error` function is employed. These are nonstandard functions in the package *dae* available from the web site http://chris.brien.name/rpackages and installed automatically in the pools where R is available. For instructions on installation see the Statistical Modelling resources web site. To obtain a description of the function, use `help(function)` in the R console.

The final stage of the analysis will be the further, detailed examination of treatment differences — either multiple comparisons analysis, the fitting of polynomial trends or, in some special cases, investigating specific contrasts. To fit polynomial trends the each quantitative `factor` must be converted to `ordered` and polynomial contrasts associated with the `ordered`. Then the `split` argument is used in the `summary` function. For example, if there were 7 treatments then the following function will fit a cubic and compute the Deviations line:

```
> summary(Experiment.aov, split = list(Treatment =
                    list(L = 1, Q = 2, C = 3, Dev = 4:6)))
```

Because there are 7 treatments the maximum degree polynomial that can be fitted is of degree 6 and so those above cubic, degrees 4 to 6, are combined into the Deviations line.

In some examples, more than one of these methods for examining treatment differences is given for illustrative purposes only. In each case, the most appropriate one should be selected for reporting the results of the analysis. Also, plots will usually be performed to illustrate the conclusions about treatment differences. Usually, bar charts (Bar Y Min Base) will be used for qualitative factors and scatter plots with fitted polynomials (Poly Fit) for quantitative factors. Note that must call `aov` without using the `Error` function for the multiple comparisons function, `multicomp`, to work.

C.3. Completely randomized design

Example II.2 Caffeine effects on students

To illustrate the procedures, I am going to use an experiment in which the effect of orally ingested caffeine on a physical task was investigated (Draper and Smith, 1981, sec.9.1). Thirty healthy male college students were selected and trained in finger tapping. Ten men were randomly assigned to receive one of three doses of caffeine (0, 100 or 200 mg). The number of finger taps after ingesting the caffeine was recorded for each student and the data were as follows:

The R expressions required to produce the complete analysis of this example are as follows:

R expressions to produce analysis

```
#set up data.frame with factors Students and Dose;
# then add response variable Taps.
CRDCaff.dat <- data.frame(Students = factor(1:30),
                          Dose = factor(rep(c(0,100,200), times=10)))
CRDCaff.dat$Taps <-
       c(242,248,246,245,246,248,244,245,250,248,247,252,247,248,248,
         248,250,250,242,247,246,244,246,248,246,243,245,242,244,250)
```

The following output shows how `CRDCaff.dat` is set up:

```
> CRDCaff.dat
```

```
   Students Dose Taps
 1        1    0  242
 2        2  100  248
 3        3  200  246
 4        4    0  245
 5        5  100  246
 6        6  200  248
 7        7    0  244
 8        8  100  245
 9        9  200  250
10       10    0  248
11       11  100  247
12       12  200  252
13       13    0  247
14       14  100  248
15       15  200  248
16       16    0  248
17       17  100  250
18       18  200  250
19       19    0  242
20       20  100  247
21       21  200  246
22       22    0  244
23       23  100  246
24       24  200  248
25       25    0  246
26       26  100  243
27       27  200  245
28       28    0  242
29       29  100  244
30       30  200  250
```

Further expressions:

```
attach(CRDCaff.dat)
#
# initial analysis
#
boxplot(split(Taps, Dose), xlab="Dose", ylab="Number of taps")
Caffeine.aov <- aov(Taps ~ Dose + Error(Students), CRDCaff.dat)
summary(Caffeine.aov)
#
# plots for diagnostic checking
#
res <- resid.errors(Caffeine.aov)
fit <- fitted.errors(Caffeine.aov)
data.frame(Students,Dose,Taps,res,fit)
plot(fit, res, pch = 16)
qqnorm(res, pch = 16)
qqline(res)
#
# multiple comparisons
#
model.tables(Caffeine.aov, type="means")
#
# fit polynomials
#
t <- 3
Dose.lev <- c(0,100,200)
CRDCaff.dat$Dose <- ordered(CRDCaff.dat$Dose, levels=Dose.lev)
contrasts(CRDCaff.dat$Dose) <- contr.poly(t, scores=Dose.lev)
contrasts(CRDCaff.dat$Dose)
Caffeine.aov <- aov(Taps ~ Dose + Error(Students), CRDCaff.dat)
summary(Caffeine.aov, split = list(Dose = list(L = 1, Q = 2)))
#
#get fitted equation
#
```

```
D <- as.vector(Dose)
D <- as.numeric(D)
Caffeine.lm <- lm(Taps ~ D)
coef(Caffeine.lm)
#
# plot means and fitted line
#
Caffeine.tab <- model.tables(Caffeine.aov, type="means")
Dose.Mean <- Caffeine.tab$tables$Dose
plot(x=Dose.lev, y=Dose.Mean, xlab="Dose", ylab="No. Taps")
Caffeine.coef <- coef(Caffeine.lm)
dosex <- seq(0, 200, 1)
Dose.Fit <- Caffeine.coef[[1]] + Caffeine.coef[[2]]*dosex
lines(x=dosex, y=Dose.Fit, type="l")
#doing quadratic regression
D2 <- D*D
Caffeine.lm <- lm(Taps ~ D + D2)
#plot means and fitted quadratic
Caffeine.tab <- model.tables(Caffeine.aov, type="means")
Dose.Mean <- Caffeine.tab$tables$Dose
plot(x=Dose.lev, y=Dose.Mean, xlab="Dose", ylab="No. Taps")
Caffeine.coef <- coef(Caffeine.lm)
dosex <- seq(0, 200, 1)
Dose.Fit <- Caffeine.coef[[1]] + Caffeine.coef[[2]]*dosex +
                                 Caffeine.coef[[3]]*dosex*dosex
lines(x=dosex, y=Dose.Fit, type="l")
```

C.4. Randomized complete block design

The example below is for a standard randomized complete block design. However, the same expressions would be used for a generalized randomized complete block design.

Example IV.1 Penicillin yield

The R expressions required to produce the complete analysis of this example are given below. They assume that the data frame `RDBDPen.dat` has been appropriately set up prior to using them. One way to set it up is as follows:

```
> attach(RDBDPen.dat)
> RDBDPen.dat
   Blend Flask Treat Yield
 1     1     1     A    89
 2     1     2     B    88
 3     1     3     C    97
 4     1     4     D    94
 5     2     1     A    84
 6     2     2     B    77
 7     2     3     C    92
 8     2     4     D    79
 9     3     1     A    81
10     3     2     B    87
11     3     3     C    87
12     3     4     D    85
13     4     1     A    87
14     4     2     B    92
15     4     3     C    89
16     4     4     D    84
17     5     1     A    79
18     5     2     B    81
19     5     3     C    80
20     5     4     D    88
```

The form of the `aov` function depends on whether Blend is fixed or random as Blend must be included outside the `Error` function if Blend is fixed, but not if it is random. Thus for Blend fixed the form of the `aov` function is

```
RCBDPen.aov <- aov(Yield ~ Blend + Treat + Error(Blend/Flask), RDBDPen.dat)
```

whereas for Blend random it is

```
RCBDPen.aov <- aov(Yield ~ Treat + Error(Blend/Flask), RDBDPen.dat)
```

R expressions to produce analysis

```
attach(RDBDPen.dat)
boxplot(split(Yield, Blend), xlab="Blend", ylab="Yield")
boxplot(split(Yield, Treat), xlab="Treatment", ylab="Yield")

RCBDPen.aov <- aov(Yield ~ Blend + Treat + Error(Blend/Flask), RDBDPen.dat)
```

or

```
RCBDPen.aov <- aov(Yield ~ Treat + Error(Blend/Flask), RDBDPen.dat)


summary(RCBDPen.aov)
#Compute Blend F and p
Blend.F <- 66/18.833
Blend.p <- 1-pf(Blend.F, 4, 12)
data.frame(Blend.F,Blend.p)
#
# Diagnostic checking
#
res <- resid.errors(RCBDPen.aov)
fit <- fitted.errors(RCBDPen.aov)
data.frame(Blend,Flask,Treat,Yield,res,fit)
plot(fit, res, pch=16)
qqnorm(res, pch=16)
qqline(res)
tukey.1df(RCBDPen.aov, RCBDPen.dat, error.term="Blend:Flask")
tukey.1df(RCBDPen.BlendRandom.aov, RCBDPen.dat, error.term="Blend:Flask")
#
# Plotting Treat means
#
RCBDPen.tab <- model.tables(RCBDPen.aov, type="means")
RCBDPen.Treat.Mean <- data.frame(Treat.lev = levels(Treat),
                                 Treat.Mean = as.vector(RCBDPen.tab$tables$Treat))
barchart(Treat.Mean ~ Treat.lev, ylim=c(0,90), xlab="Treatment",
         ylab="Yield (%)", main="Fitted values for Yield",
         data=RCBDPen.Treat.Mean)
```

C.5. Latin square design

Example V.2 Pollution effects of petrol additives

The R expressions required to produce the complete analysis of this example are given below. They assume that the data frame `LSPolut.dat` has been appropriately set up prior to using them. One way to set it up is as follows:

```
> load("LSPolut.dat.rda")
> attach(LSPolut.dat)
> LSPolut.dat
   Units Permutation Drivers Cars Additives Reduct.NO
1      1          11       1    1         B        20
2      2          12       1    2         D        20
3      3          10       1    3         C        17
4      4           9       1    4         A        15
5      5           7       2    1         A        20
6      6           8       2    2         B        27
7      7           6       2    3         D        23
8      8           5       2    4         C        26
9      9          15       3    1         D        20
10    10          16       3    2         C        25
11    11          14       3    3         A        21
12    12          13       3    4         B        26
13    13           3       4    1         C        16
14    14           4       4    2         A        16
15    15           2       4    3         B        15
16    16           1       4    4         D        13
>
```

Again the form of the `aov` function depends on whether Drivers and Cars are fixed or random. Delete the random ones from outside the `Error` function. For example if both are random the call to `aov` would become

```
LSPolut.aov <- aov(Reduct.NO ~ Additives + Error(Drivers*Cars), LSPolut.dat)
```

R expressions to produce analysis

```
boxplot(split(Reduct.NO, Drivers), xlab="Drivers", ylab="Reduction in NO")
boxplot(split(Reduct.NO, Cars), xlab="Cars", ylab="Reduction in NO")
boxplot(split(Reduct.NO, Additives), xlab="Additives", ylab="Reduction in NO")
LSPolut.aov <- aov(Reduct.NO ~ Drivers + Cars + Additives + Error(Drivers*Cars),
LSPolut.dat)
summary(LSPolut.aov)
#Compute Drivers and Cars Fs and p-values
Drivers.F <- 72/2.667
Drivers.p <- 1-pf(Drivers.F, 3, 6)
Cars.F <- 8/2.667
Cars.p <- 1-pf(Cars.F, 3, 6)
data.frame(Drivers.F,Drivers.p,Cars.F,Cars.p)
#
# Diagnostic checking
#
res <- resid.errors(LSPolut.aov)
fit <- fitted.errors(LSPolut.aov)
data.frame(Drivers,Cars,Additives,Reduct.NO,res,fit)
plot(fit, res, pch=16)
qqnorm(res, pch=16)
qqline(res)
tukey.1df(LSPolut.aov, LSPolut.dat, error.term = "Drivers:Cars")
```

```
#
# multiple comparisons
#
model.tables(LSPolut.aov, type="means")
q <- qtukey(0.95, 4, 6)
q
#
# Plotting Treat means
#
LSPolut.tab <- model.tables(LSPolut.aov, type="means")
LSPolut.Adds.Mean <- data.frame(Adds.lev = levels(Additives),
                        Adds.Mean = as.vector(LSPolut.tab$tables$Additives))
LSPolut.Adds.Mean <- LSPolut.Adds.Mean[order(LSPolut.Adds.Mean$Adds.Mean),]
#use factor to order bars
LSPolut.Adds.Mean$Adds.lev <- factor(LSPolut.Adds.Mean$Adds.lev,
                                     levels=LSPolut.Adds.Mean$Adds.lev)
barchart(Adds.Mean ~ Adds.lev, xlab="Additives", ylim=c(0,25),
         ylab="NO Reduction", main="Fitted values for Nitrous Oxide Reduction",
         data=LSPolut.Adds.Mean)
```

C.6. A set of Latin squares design

In general, the analysis for a set of the Latin squares is obtained by using a *model formula* in which the explanatory variables are specified to be

randomized structure formula + Error(unrandomized structure formula)

In addition, fixed terms in the unrandomized structure formula need to be also included with the terms outside the `Error` function, as mentioned in the introduction to this Appendix.

Example Case 2 — same Cars different Drivers

As an example of an experiment involving a set of Latin squares, we illustrate the analysis for the case in which petrol additives are assigned to the combinations of four drivers and four cars using a Latin square and this is done for two different occasions with same 4 cars on both occasions, but with the drivers on one occasion unconnected with those on the other. As a result the rows of the square, but not the columns, are rerandomized on the second occasion.

The R expressions required to produce the complete analysis of this example are given below. They assume that the data window `LSRepeat2.dat` has been appropriately set up prior to using them. One way to set it up follows — note that Data is randomly generated data to enable an analysis to be performed.

```
> LSRepeat2.dat <- data.frame(LSRepeat2.lay, Data = rnorm(n))
> attach(LSRepeat2.dat)
> LSRepeat2.dat
   Units Permutation Occasion Drivers Cars Additives        Data
1      1          21        1       1    1         C  1.33299051
2      2          24        1       1    2         A  1.52009255
3      3          22        1       1    3         B  0.18680922
4      4          23        1       1    4         D  0.26281501
5      5          17        1       2    1         A -0.07712774
6      6          20        1       2    2         C -1.31362565
7      7          18        1       2    3         D -0.71681803
8      8          19        1       2    4         B  0.16637414
9      9          29        1       3    1         B -0.59372326
10    10          32        1       3    2         D -0.76442376
11    11          30        1       3    3         C  0.77602428
12    12          31        1       3    4         A  0.66442845
13    13          25        1       4    1         D  0.90195336
14    14          28        1       4    2         B  0.12874893
15    15          26        1       4    3         A -0.68993861
16    16          27        1       4    4         C -0.76842248
17    17           5        2       1    1         D  0.41973900
18    18           8        2       1    2         B -0.39102629
19    19           6        2       1    3         A -0.41841197
20    20           7        2       1    4         C -0.24032621
21    21          13        2       2    1         A -0.28418206
22    22          16        2       2    2         C -0.25538454
23    23          14        2       2    3         D -0.71548353
24    24          15        2       2    4         B  1.01486123
25    25           9        2       3    1         C -0.41233805
26    26          12        2       3    2         A -1.23423748
27    27          10        2       3    3         B  0.26930868
28    28          11        2       3    4         D  0.99496028
29    29           1        2       4    1         B  1.90147149
30    30           4        2       4    2         D  0.49720016
31    31           2        2       4    3         C  0.08308735
32    32           3        2       4    4         A  2.00740238
```

Again the form of the `aov` function depends on whether Occasion, Drivers and Cars are fixed or random — the expressions below contain the `aov` function if all terms except Drivers:Cars[Occasion] are fixed. If some of Occasion, Drivers and Cars are random, delete the random factors from outside the `Error` function, except that Occasion can only be deleted if Drivers is also random. For example, if all three factors are random, the call to `aov` would become

```
LSRepeat2.aov <- aov(Data ~ Additives + Error((Occasion/Drivers) * Cars),
                     LSRepeat2.dat)
```

R expressions to produce analysis

```
boxplot(split(Data, Occasion), xlab="Occasion", ylab="Data")
boxplot(split(Data, Drivers), xlab="Drivers", ylab="Data")
boxplot(split(Data, Cars), xlab="Cars", ylab="Data")
boxplot(split(Data, Additives), xlab="Additives", ylab="Data")
LSRepeat2.aov <- aov(Data ~ Occasion/Drivers + Occasion*Cars + Additives +
                                    Error((Occasion/Drivers)*Cars), LSRepeat2.dat)
summary(LSRepeat2.aov)
#
# Diagnostic checking
#
res <- resid.errors(LSRepeat2.aov)
fit <- fitted.errors(LSRepeat2.aov)
plot(fit, res, pch=16)
qqnorm(res, pch=16)
qqline(res)
tukey.1df(LSRepeat2.aov, LSRepeat2.dat, error.term = "Occasion:Drivers:Cars")
#
# multiple comparisons
#
model.tables(LSRepeat2.aov, type="means")
q <- qtukey(0.95, 4, 15)
q
#
# Plotting Treat means
#
LSRepeat2.tab <- model.tables(LSRepeat2.aov, type="means")
LSRepeat2.Adds.Mean <- data.frame(Adds.lev = levels(Additives),
                        Adds.Mean = as.vector(LSRepeat2.tab$tables$Additives))
LSRepeat2.Adds.Mean <- LSRepeat2.Adds.Mean[order(LSRepeat2.Adds.Mean$Adds.Mean),]
#use factor to order bars
LSRepeat2.Adds.Mean$Adds.lev <- factor(LSRepeat2.Adds.Mean$Adds.lev,
                                    levels=LSRepeat2.Adds.Mean$Adds.lev)
barchart(Adds.Mean ~ Adds.lev, xlab="Additives",
         ylab="NO Reduction", main="Fitted values for Nitrous Oxide Reduction",
         data=LSRepeat2.Adds.Mean)
```

C.7. Factorial experiments

Layouts for factorial experiments can be obtained in R using the expressions for the chosen design when only a single-factor is involved. The difference with factorial experiments is that the several treatment factors are entered in standard order.

Initial graphical exploration for factorial experiments utilises interaction plots, produced with the function `interaction.plot` for two factors or the nonstandard function `interaction.ABC.plot`, available from the *dae* library, for more than two factors. We do not produce boxplots in this case, in part because they look at just the overall effects of one factor and are only relevant if the factors are independent.

Note that for factorial experiments the model formula used in the `aov` function should involve the terms arising from the design employed (Blocks for the RCBD and Rows + Columns for the Latin Square) as well as the treatment factors (for example A * B * C).

As usual, multiple comparisons procedures can be employed and R used to obtain the tables of means and studentized ranges needed in the calculations.

Fitting polynomial submodels for factorial experiments in R is an extension of the procedure for a single factor described in the introduction to this appendix. You have to a) specify the polynomial contrasts for each quantitative factor, b) use the crossed (*) operator between factors in the model formula (for example A * B), and c) specify all the terms involving the quantitative factors in the `list` argument of the `summary` function. The general form of the `summary` function for one factor, B say, quantitative is:

```
summary(Experiment.aov,
       split = list(B = list(L = 1, Q = 2, Dev = 3:(b-1)),
                 "A:B" = list(L = 1, Q = 2, Dev = 3:(b-1))))
```

and for two factors, A and B say, quantitative is

```
summary(Experiment.aov,
        split = list(A = list(L = 1, Q = 2, Dev = 3:(a-1)),
                     B = list(L = 1, Q = 2, Dev = 3:(b-1)),
                  "A:B" = list(L.L=1, L.Q=2, Q.L=b, Q.Q=(b+1),
                               Dev=c(3:(b-1),(b+2:(a-1)(b-1))))))
```

Note the use of `Dev` to collect together the remaining degrees of freedom. Of course, if *a* or *b* is 3, the corresponding `Dev` term(s) should be omitted. Also, there is no point in using the split function for a factor with 2 levels. Note that quotation marks are needed for the terms involving two or more factors.

The labeling of terms in the case of two quantitative factors is in standard order for the two factors. You can think of the labels in a two-way table, numbered 1 to $(a-1)(b-1)$ by rows, as follows:

Factor			B				
	Contrast	Contrast Label	1 L	2 Q	3 Dev	…	(b – 1) Dev
	1	L	L.L	L.Q	Dev	…	Dev
	2	Q	Q.L	Q.Q	Dev	…	Dev
A	3	Dev	Dev	Dev	Dev	…	Dev
	(a – 1)	Dev	Dev	Dev	Dev	…	Dev

As for the single-factor case, the fitted equation is obtained using the `lm` function where the independent variables are the values of the quantitative factors and appropriate powers and products, saved in numeric vectors prior to invoking `lm`. Thus if `R.Fac` and `S.Fac` are two quantitative factors, then expressions for obtaining terms for a model involving all possible terms for polynomials up to order 2 and the associated `lm` function are:

```
R <- as.numeric(as.vector(R.Fac))
S <- as.numeric(as.vector(S.Fac))
R2 <- R*R; S2 < S*S
R1S1 <- R*S; R1S2 <- R*S2; R2S1 <- R2*S; R2S2 <- R2*S2
```

```
lm(y ~ R + R2 + S + S2 + R1S1 + R1S2 + R2S1 +R2S2, data.frame)
```

You will need to delete terms deemed unnecessary by the hypothesis testing. To plot the fitted surface use commands of the following form:

```
Experiment.lm <- lm(y ~ poly(R, u) + poly(S, v) + poly(R, w) *
   poly(S, x), singular.ok=T)
Experiment.grid <- list(R = seq(min(R), max(R), length = 40),
                        S = seq(min(S), max(S), length = 40))
Experiment.surf <- expand.grid(Experiment.grid)
Experiment.surf$y <- as.vector(predict(Experiment.lm,
   Experiment.surf))
wireframe(y ~ R*S, data= Experiment.surf, drape=TRUE)
```

You will need to substitute new names for *Experiment*, *y*, *R*, *S*, *u*, *v*, *w*, *x*. Note that *u* and *v* are the orders of the main-effect terms and *w* and *x* are the maximum orders of each factor in cross-product terms. See the practical for this chapter for examples.

Example VII.5 Animal survival experiment

There are many ways to set up the `data.frame` for a factorial expressions. One way to do this for the example is given by the following R expressions that enter it into the `data.frame Fac2Pois.dat`:

```
> Fac2Pois.dat <- fac.gen(generate = list(Poison = 3, 4, Treat=4))
> Fac2Pois.dat <- data.frame(Animals = factor(1:48), Fac2Pois.dat)
> Fac2Pois.dat$Surv.Time <-
+                   c(0.31,0.82,0.43,0.45,0.45,1.10,0.45,0.71,0.46,0.88,0.63,0.66,
+                     0.43,0.72,0.76,0.62,0.36,0.92,0.44,0.56,0.29,0.61,0.35,1.02,
+                     0.40,0.49,0.31,0.71,0.23,1.24,0.40,0.38,0.22,0.30,0.23,0.30,
+                     0.21,0.37,0.25,0.36,0.18,0.38,0.24,0.31,0.23,0.29,0.22,0.33)
> attach(Fac2Pois.dat)
> Fac2Pois.dat
   Animals Poison Treat Surv.Time
 1       1      1     1      0.31
 2       2      1     2      0.82
 3       3      1     3      0.43
 4       4      1     4      0.45
 5       5      1     1      0.45
 6       6      1     2      1.10
 7       7      1     3      0.45
 8       8      1     4      0.71
 9       9      1     1      0.46
10      10      1     2      0.88
11      11      1     3      0.63
12      12      1     4      0.66
13      13      1     1      0.43
14      14      1     2      0.72
15      15      1     3      0.76
16      16      1     4      0.62
17      17      2     1      0.36
18      18      2     2      0.92
19      19      2     3      0.44
20      20      2     4      0.56
21      21      2     1      0.29
22      22      2     2      0.61
23      23      2     3      0.35
24      24      2     4      1.02
25      25      2     1      0.40
```

```
26      26      2     2      0.49
27      27      2     3      0.31
28      28      2     4      0.71
29      29      2     1      0.23
30      30      2     2      1.24
31      31      2     3      0.40
32      32      2     4      0.38
33      33      3     1      0.22
34      34      3     2      0.30
35      35      3     3      0.23
36      36      3     4      0.30
37      37      3     1      0.21
38      38      3     2      0.37
39      39      3     3      0.25
40      40      3     4      0.36
41      41      3     1      0.18
42      42      3     2      0.38
43      43      3     3      0.24
44      44      3     4      0.31
45      45      3     1      0.23
46      46      3     2      0.29
47      47      3     3      0.22
48      48      3     4      0.33
```

The R expressions required to produce the complete analysis of this example are as follows.

R expressions to produce analysis

```
attach(Fac2Pois.dat)
Fac2Pois.dat
interaction.plot(Poison, Treat, Surv.Time, lwd=4)
boxplot(split(Surv.Time, Poison), xlab="Poison", ylab="Survival time (10 hours)")
boxplot(split(Surv.Time, Treat), xlab="Treatment", ylab="Survival time (10 hours)")
Fac2Pois.aov <- aov(Surv.Time ~ Poison * Treat + Error(Animals), Fac2Pois.dat)
summary(Fac2Pois.aov)
#
# Diagnostic checking
#
res <- resid.errors(Fac2Pois.aov)
fit <- fitted.errors(Fac2Pois.aov)
data.frame(Animals,Poison,Treat,Surv.Time,res,fit)
plot(fit, res, pch=16)
plot(as.numeric(Poison), res, pch=16)
plot(as.numeric(Treat), res, pch=16)
qqnorm(res, pch=16)
qqline(res)
Fac2Pois.NoError.aov <- aov(Surv.Time ~ Poison * Treat, Fac2Pois.dat)
library(MASS)
boxcox(Fac2Pois.NoError.aov, lambda=seq(from = -2.5, to = 2.5, len=20), plotit=T)
#
# re-analysis
#
detach(Fac2Pois.dat)
Fac2Pois.dat$Death.Rate <- 1/Fac2Pois.dat$Surv.Time
attach(Fac2Pois.dat)
interaction.plot(Poison, Treat, Death.Rate, lwd=4)
Fac2Pois.DR.aov <- aov(Death.Rate ~ Poison * Treat + Error(Animals), Fac2Pois.dat)
summary(Fac2Pois.DR.aov)
res <- resid.errors(Fac2Pois.DR.aov)
fit <- fitted.errors(Fac2Pois.DR.aov)
plot(fit, res, pch=16)
plot(as.numeric(Poison), res, pch=16)
plot(as.numeric(Treat), res, pch=16)
qqnorm(res, pch=16)
qqline(res)
#
```

```
# multiple comparisons
#
model.tables(Fac2Pois.DR.aov, type="means")
q.PT <- qtukey(0.95, 12, 36)
q.PT
q.P <- qtukey(0.95, 3, 36)
q.P
q.T <- qtukey(0.95, 4, 36)
q.T
#
# Plotting means
#
Fac2Pois.DR.tab <- model.tables(Fac2Pois.DR.aov, type="means")
Fac2Pois.DR.Poison.Means <-
              data.frame(Poison = levels(Poison),
                         Death.Rate = as.vector(Fac2Pois.DR.tab$tables$Poison))
barchart(Death.Rate ~ Poison, main="Fitted values for Death rate", ylim=c(0,4),
              data=Fac2Pois.DR.Poison.Means)
Fac2Pois.DR.Treat.Means <-
              data.frame(Treatment = levels(Treat),
                         Death.Rate = as.vector(Fac2Pois.DR.tab$tables$Treat))
barchart(Death.Rate ~ Treat, main="Fitted values for Death rate", ylim=c(0,4),
              data=Fac2Pois.DR.Treat.Means)
```

Example C.1 Grafting experiment

The R expressions required to produce the complete analysis of this example are as follows. They assume that the Data window `Fac2Take.dat` has been appropriately set up prior to using them. One way to set it up is as follows:

```
> attach(Fac2Take.dat)
> Fac2Take.dat
   Blocks Plots A B Take Cell.1.1 Treats
1       1     1 1 1   64        1      1
2       1     2 2 1   23        2      3
3       1     3 1 2   30        2      2
4       1     4 2 2   15        2      4
5       2     1 1 1   75        1      1
6       2     2 2 1   14        2      3
7       2     3 1 2   50        2      2
8       2     4 2 2   33        2      4
9       3     1 1 1   76        1      1
10      3     2 2 1   12        2      3
11      3     3 1 2   41        2      2
12      3     4 2 2   17        2      4
13      4     1 1 1   73        1      1
14      4     2 2 1   33        2      3
15      4     3 1 2   25        2      2
16      4     4 2 2   10        2      4
```

Note that if Blocks are random the call to `aov` would become

```
    Fac2Take.aov <- aov(Take ~ A * B + Error(Blocks/Plots), Fac2Take.dat)
```

R expressions to produce analysis

```
attach(Fac2Take.dat)
Fac2Take.dat
interaction.plot(A, B, Take, lwd=4)
Fac2Take.aov <- aov(Take ~ Blocks + A * B + Error(Blocks/Plots), Fac2Take.dat)
summary(Fac2Take.aov)
#
# recompute for missing value
#
MSq <- c(73.729, 4795.6, 1387.6, 1139.1, 2.8797)
Res <- c(rep(819.6/8, 4), 816.6828/7)
df.num <- c(3,rep(1,4))
df.den <- c(rep(8, 4),7)
Fvalue <- MSq/Res
pvalue <- 1-pf(Fvalue, df.num, df.den)
data.frame(MSq,Res,df.num,df.den,Fvalue,pvalue)
#
# Diagnostic checking
#
res <- resid.errors(Fac2Take.aov)
fit <- fitted.errors(Fac2Take.aov)
data.frame(Blocks,Plots,A,B,Take,res,fit)
plot(fit, res, pch=16)
plot(as.numeric(A), res, pch=16)
plot(as.numeric(B), res, pch=16)
qqnorm(res, pch=16)
qqline(res)
tukey.1df(Fac2Take.aov, Fac2Take.dat, error.term = "Blocks:Plots")
#
# multiple comparisons
#
Fac2Take.tab <- model.tables(Fac2Take.aov, type="means")
Fac2Take.tab$tables$"A:B"
q <- qtukey(0.95, 4, 8)
q
#
# reanalysis for one-cell interaction model
#
Fac2Take.dat$Cell.1.1 <- factor(1 + as.numeric(A != "1" | B != "1"))
Fac2Take.dat$Treats <- fac.combine(list(A, B))
detach(Fac2Take.dat)
attach(Fac2Take.dat)
Fac2Take.dat
Fac2Take.aov <- aov(Take ~ Blocks + Cell.1.1/Treats + Error(Blocks/Plots),
Fac2Take.dat)
summary(Fac2Take.aov)
#
# recompute for missing value
#
MSq <- c(73.729,6556.7,382.8)
Res <- rep(819.6/8, 3)
df.num <- c(3, 1, 2)
Fvalue <- MSq/Res
pvalue <- 1-pf(Fvalue, df.num, 8)
data.frame(MSq,Res,df.num,Fvalue,pvalue)
```

C.8. Two-level factorial experiments

a) Replicated two-level factorial experiments

Example VIII.1 2^3 pilot plant experiment (continued)

The R expressions required to produce the complete analysis of this example are as follows.

```
#
#obtain randomized layout
#
n <- 16
mp <- c("-", "+")
Fac3Pilot.ran <- fac.gen(generate = list(Te = mp, C = mp, K = mp), each = 2,
                         order="yates")
Fac3Pilot.unit <- list(Tests = n)
Fac3Pilot.lay <- fac.layout(unrandomized = Fac3Pilot.unit,
                            randomized = Fac3Pilot.ran, seed = 897)
#sort treats into Yates order
Fac3Pilot.lay <- Fac3Pilot.lay[Fac3Pilot.lay$Permutation,]
#add Yield
Fac3Pilot.dat <- data.frame(Fac3Pilot.lay,
                            Yield = c(59, 61, 74, 70, 50, 58, 69, 67,
                                      50, 54, 81, 85, 46, 44, 79, 81))
#re-sort into randomized order
Fac3Pilot.dat <- Fac3Pilot.dat[Fac3Pilot.dat$Units,]
attach(Fac3Pilot.dat)
interaction.ABC.plot(Yield, Te, C, K, data=Fac3Pilot.dat,
                     title="Effect of Temperature(Te), Concentration(C) and
   Catalyst(K) on Yield")
Fac3Pilot.aov <- aov(Yield ~ Te * C * K + Error(Tests), Fac3Pilot.dat)
summary(Fac3Pilot.aov)
round(yates.effects(Fac3Pilot.aov, error.term = "Tests", data=Fac3Pilot.dat), 2)
#
# Diagnostic checking
#
res <- resid.errors(Fac3Pilot.aov)
fit <- fitted.errors(Fac3Pilot.aov)
plot(fit, res, pch=16)
plot(as.numeric(Te), res, pch=16)
plot(as.numeric(C), res, pch=16)
plot(as.numeric(K), res, pch=16)
qqnorm(res, pch=16)
qqline(res)
#
# treatment differences
#
Fac3Pilot.means <- model.tables(Fac3Pilot.aov, type="means")
Fac3Pilot.means$tables$"Grand mean"
Fac3Pilot.means$tables$"Te:K"
Fac3Pilot.means$tables$"C"
q <- qtukey(0.95, 4, 8)
q
```

Note the use of the `round` function with the `yates.effects` function to obtain nicer output by rounding the effects to 2 decimal places.

b) Unreplicated two-level factorial experiments

The analysis of single replicate of a 2^k factorial experiments is based the normal probability plot of Yates effects, followed by an ANOVA for fitted model so that tables of means and residuals can be obtained and diagnostic checking performed.

Example VIII.2 A 2^4 process development study (continued)

The R expressions required to produce the complete analysis of this example are as follows.

```
#
# set up data frame and obtain initial analysis
#
mp <- c("-", "+")
fnames <- list(Catal = mp, Temp = mp, Press = mp, Conc = mp)
Fac4Proc.Treats <- fac.gen(generate = fnames, order="yates")
Fac4Proc.dat <- data.frame(Runs = factor(1:16), Fac4Proc.Treats)
remove("Fac4Proc.Treats")
Fac4Proc.dat$Conv <- c(71,61,90,82,68,61,87,80,61,50,89,83,59,51,85,78)
attach(Fac4Proc.dat)
Fac4Proc.dat
Fac4Proc.aov <- aov(Conv ~ Catal * Temp * Press * Conc + Error(Runs),
                                                                Fac4Proc.dat)
summary(Fac4Proc.aov)
round(yates.effects(Fac4Proc.aov, error.term="Runs", data=Fac4Proc.dat), 2)
# Perform analysis assuming 3- & 4-factor interactions negligible
Fac4Proc.TwoFac.aov <- aov(Conv ~ (Catal + Temp + Press + Conc)^2 + Error(Runs),
                                                                Fac4Proc.dat)
summary(Fac4Proc.TwoFac.aov)
#
#Yates effects probability plot
#
qqyeffects(Fac4Proc.aov, error.term="Runs", data=Fac4Proc.dat)
#
# Diagnostic checking
#
Fac4Proc.Fit.aov <- aov(Conv ~ Temp * Conc + Catal + Press + Error(Runs),
Fac4Proc.dat)
summary(Fac4Proc.Fit.aov)
tukey.1df(Fac4Proc.Fit.aov, Fac4Proc.dat, error.term="Runs")
res <- resid.errors(Fac4Proc.Fit.aov)
fit <- fitted.errors(Fac4Proc.Fit.aov)
plot(fit, res, pch=16)
qqnorm(res, pch=16)
qqline(res)
plot(as.numeric(Temp), res, pch=16)
plot(as.numeric(Conc), res, pch=16)
plot(as.numeric(Catal), res, pch=16)
plot(as.numeric(Press), res, pch=16)
#
# treatment differences
#
Fac4Proc.means <- model.tables(Fac4Proc.aov, type="means")
Fac4Proc.means$tables$"Grand mean"
Fac4Proc.means$tables$"Temp:Conc"
Fac4Proc.means$tables$"Catal"
Fac4Proc.means$tables$"Press"
interaction.plot(Temp, Conc, Conv)
q <- qtukey(0.95, 4, 10)
q
```

c) Confounded two-level factorial experiments

In general, the analysis of confounded designs depends on whether only a single replicate of the treatments has been observed in the experiment or several replicates of the complete set of treatments have been observed.

If a single replicate of the treatments has been observed, then the analysis must be based on the normal probability plot of the Yates effects. On the other hand, if two or more complete sets of treatments have been observed, the analysis can be based on an analysis of variance table. To get the correct analysis, you must use an `Error` function as part of the model formula.

Example VIII.7 Partial confounding in a repeated four block experiment (continued)

As four complete sets of treatments have been observed, the analysis will use an analysis of variance table. Having set up the factors and data in an R `data.frame`, the instructions to produce this analysis are as follows:

```
attach(Fac3Conf.4Blocks.Partial.dat)
Fac3Conf.4Blocks.Partial.dat
Fac3Conf.4Blocks.Partial.aov <- aov(Yield ~ A * B * C + Error(Blends/Runs),
   Fac3Conf.4Blocks.Partial.dat)
summary(Fac3Conf.4Blocks.Partial.aov)
#
# Diagnostic checking
#
tukey.1df(Fac3Conf.4Blocks.Partial.aov, Fac3Conf.4Blocks.Partial.dat,
   error.term="Runs %in% Blends")
res <- resid.errors(Fac3Conf.4Blocks.Partial.aov)
fit <- fitted.errors(Fac3Conf.4Blocks.Partial.aov)
plot(fit, res, pch=16)
qqnorm(res, pch=16)
qqline(res)
plot(as.numeric(A), res, pch=16)
plot(as.numeric(B), res, pch=16)
plot(as.numeric(C), res, pch=16)
```

Note the inclusion of `Error(Blends/Runs)` in the model formula.

d) Fractional two-level factorial experiments

Generally the analysis of fractional 2^k designs is based the normal probability plot of Yates effects, followed by an ANOVA for fitted model so that tables of means and residuals can be obtained and diagnostic checking performed.

Example VIII.10 A bike experiment

The following expressions are used to analyse the first fraction:

```
#
# set up data.frame
#
mp <- c("-", "+")
fnames <- list(Seat = mp, Dynamo = mp, Handbars = mp)
Frf7Bike.Treats <- fac.gen(generate = fnames, order="yates")
```

```
attach(Frf7Bike.Treats)
Frf7Bike.Treats$Gear <- factor(mpone(Seat)*mpone(Dynamo), labels = mp)
Frf7Bike.Treats$Raincoat <- factor(mpone(Seat)*mpone(Handbars), labels = mp)
Frf7Bike.Treats$Brekkie <- factor(mpone(Dynamo)*mpone(Handbars), labels = mp)
Frf7Bike.Treats$Tyres <- factor(mpone(Seat)*mpone(Dynamo)*mpone(Handbars),
                                                                labels = mp)
detach(Frf7Bike.Treats)
Frf7Bike.dat <- data.frame(Runs = factor(1:8), Frf7Bike.Treats)
Frf7Bike.dat$Time <- as.vector(c(69, 52, 60, 83, 71, 50, 59, 88))
Frf7Bike.dat
#
# analyse
#
Frf7Bike.aov <- aov(Time ~ (Seat + Dynamo + Handbars + Gear + Raincoat + Brekkie +
                    Tyres)^2 + Error(Runs), Frf7Bike.dat)
summary(Frf7Bike.aov)
qqyeffects(Frf7Bike.aov, error.term = "Runs", data=Frf7Bike.dat)
round(yates.effects(Frf7Bike.aov, error.term="Runs", data=Frf7Bike.dat), 2)
```

Note the use of '+' and `^2` in the model formula for `aov` so that the analysis only takes into account main effects and two-factor interactions. Diagnostic checking is not performed in this case because of the very low residual degrees of freedom.

The following expressions are used to combine the two fractions and analyse the combined data.

```
#
# combine fractions
#
Frf7Bike.Both.dat <- rbind(Frf7Bike.dat,Frf7Bike2.dat)
Frf7Bike.Both.dat <- data.frame(Block = factor(rep(1:2, each=8)),
                                Frf7Bike.Both.dat)
Frf7Bike.Both.dat
#
# analyse
#
Frf7Bike.Both.aov <- aov(Time ~ Block + (Seat + Dynamo + Handbars + Gear +
                                Raincoat + Brekkie + Tyres)^2 +
                                Error(Block/Runs), Frf7Bike.Both.dat)
summary(Frf7Bike.Both.aov)
qqyeffects(Frf7Bike.Both.aov, error.term = "Block:Runs",
               data=Frf7Bike.Both.dat)
round(yates.effects(Frf7Bike.Both.aov, error.term="Block:Runs",
                    data=Frf7Bike.Both.dat), 2)
#
# re-do analysis for just fitted model followed by diagnostic checking
#
Frf7Bike.Both.Fit.aov <- aov(Time ~ Block + Dynamo + Gear + Error(Block/Runs),
Frf7Bike.Both.dat)
summary(Frf7Bike.Both.Fit.aov)
tukey.1df(Frf7Bike.Both.Fit.aov, Frf7Bike.Both.dat, error.term="Block:Runs")
res <- resid.errors(Frf7Bike.Both.Fit.aov)
fit <- fitted.errors(Frf7Bike.Both.Fit.aov)
plot(fit, res, pch=16)
qqnorm(res, pch=16)
qqline(res)
plot(as.numeric(Frf7Bike.Both.dat$Seat), res, pch=16)
plot(as.numeric(Frf7Bike.Both.dat$Dynamo), res, pch=16)
plot(as.numeric(Frf7Bike.Both.dat$Handbars), res, pch=16)
plot(as.numeric(Frf7Bike.Both.dat$Gear), res, pch=16)
plot(as.numeric(Frf7Bike.Both.dat$Raincoat), res, pch=16)
plot(as.numeric(Frf7Bike.Both.dat$Brekkie), res, pch=16)
plot(as.numeric(Frf7Bike.Both.dat$Tyres), res, pch=16)
```

Again '+' and `^2` are used in the model formula for `aov` to restrict the analysis to main effects and two-factor interactions.

C.9. Split-plot experiment

Example IX.1 Production rate experiment

```
attach(SPLProd.dat)
interaction.plot(Methods, Sources, Prodn, lwd = 4)
SPLProd.aov <- aov(Prodn ~ Factories + Methods * Sources +
                          Error(Factories/Areas/Parts), SPLProd.dat)
summary(SPLProd.aov)
#Compute Factories and Areas[Factories] Fs and p-values
Factories.F <- 424.07/315.7
Factories.p <- 1-pf(Factories.F, 3, 6)
Factories.Areas.F <- 315.7/136.94
Factories.Areas.p <- 1-pf(Factories.Areas.F, 6, 18)
data.frame(Factories.F,Factories.p,Factories.Areas.F,Factories.Areas.p)
#
# Diagnostic checking
#
tukey.1df(SPLProd.aov, SPLProd.dat, "Factories:Areas:Parts")
res <- resid.errors(SPLProd.aov)
fit <- fitted.errors(SPLProd.aov)
plot(fit, res, pch=16)
qqnorm(res, pch=16)
qqline(res)
plot(as.numeric(Methods), res, pch=16)
plot(as.numeric(Sources), res, pch=16)
#
# tables of means
#
SPLProd.means <- model.tables(SPLProd.aov, type="means")
SPLProd.means
qtukey(0.95, 3, 6)
qtukey(0.95, 3, 18)
```

Note that Factories occurs both inside and outside the `Error` function in the model formula used in the `aov` function for analysing this experiment. This is because Factories has been designated as a fixed, unrandomized factor. Also, the F-ratios for Factories and Areas[Factories] have to be computed explicitly because they are not given in the output from the `summary` function.