Exercises for Day 6 – Solution
Applied Statistics & Statistical methods for SCIENCE

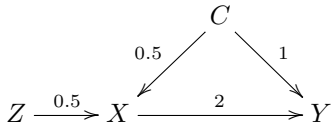
Anton Rask Lundborg

January 2024

Exercise 6.1 Causal inference using an instrumental variable

Problem

The purpose of this exercise is to try the *two-step procedure* using an instrumental variable. In order to know the true value of the parameters we simulate a dataset from the following model:



Here the regression coefficients are written above the arrows. Furthermore, normal errors with unit standard deviation are added to all variables. Simulating $N = 100$ samples from this model may be done via the R code:

```
N <- 100
C <- rnorm(N)
Z <- rnorm(N)
X <- 0.5 * Z + 0.5 * C + rnorm(N)
Y <- 2 * X + C + rnorm(N)
```

We wish to recover the causal effect of X on Y (i.e. the coefficient 2) from the dataset, where only Z, X and Y are available. Thus, the *confounder* C is not available, but we have the *instrumental variable* Z at our disposal. Answer the following questions:

- Simulate the dataset in R using the above code.
- Fit a simple linear regression of Y on X. Do you recover the coefficient 2 from this regression? Why not? Why is the estimate biased to return too large a value?
- Implement the *two-step procedure* described on lecture slide 17 (Day 6). Hint: To find the predicted values for X given the instrumental variable Z you may use the code

  ```
  hatX <- predict(lm(X ~ Z))
  ```

- Do you recover the coefficient 2 from the *two-step procedure*? Is this still true if you simulate a new dataset several times?
- Try to see if you can recover the coefficient 2 when the sample size equals $N = 10000$. What does this say about the power of the *two-step procedure*?
- Now assume that the confounder C is available. Can you recover the true causal effect of X on Y from $N = 100$ observations?

Solution

We simulate the data and fit a linear regression:

```
N <- 100
C <- rnorm(N)
Z <- rnorm(N)
X <- 0.5 * Z + 0.5 * C + rnorm(N)
Y <- 2 * X + C + rnorm(N)

m <- lm(Y ~ X)
summary(m)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3241 -0.6852  0.0620  0.8809  2.7435
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.009202   0.126330   0.073    0.942
## X           2.228157   0.105461  21.128   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.261 on 98 degrees of freedom
## Multiple R-squared:   0.82,  Adjusted R-squared:  0.8181
## F-statistic: 446.4 on 1 and 98 DF,  p-value: < 2.2e-16
```

We see that the coefficient is estimated incorrectly. We can repeat the experiment 1000 times to get an idea of whether we were unlucky or whether there is a consistent bias in the estimation:

```
reps <- 1000
beta <- numeric(reps)
for (i in 1:reps) {
  N <- 100
  C <- rnorm(N)
  Z <- rnorm(N)
  X <- 0.5 * Z + 0.5 * C + rnorm(N)
  Y <- 2 * X + C + rnorm(N)

  m <- lm(Y ~ X)
  beta[i] <- coef(m)[2]
}
mean(beta)
```

```
## [1] 2.330342
```

The average value of the coefficient is significantly larger than 2. The value is too large since part of `C` is included in both the specification of `X` and of `Y` but we do not account for this. We repeat the experiment with the two-step procedure instead:

```
reps <- 1000
beta <- numeric(reps)
for (i in 1:reps) {
```

```
  N <- 100
  C <- rnorm(N)
  Z <- rnorm(N)
  X <- 0.5 * Z + 0.5 * C + rnorm(N)
  Y <- 2 * X + C + rnorm(N)

  hat_X <- predict(lm(X ~ Z))
  m <- lm(Y ~ hat_X)
  beta[i] <- coef(m)[2]
}
mean(beta)
```

```
## [1] 1.989348
```

We see that on average we get a value much closer to 2 than before, however, now we slightly underestimate the effect on average. We can up the sample size to 10000:

```
reps <- 1000
beta <- numeric(reps)
for (i in 1:reps) {
  N <- 10000
  C <- rnorm(N)
  Z <- rnorm(N)
  X <- 0.5 * Z + 0.5 * C + rnorm(N)
  Y <- 2 * X + C + rnorm(N)

  hat_X <- predict(lm(X ~ Z))
  m <- lm(Y ~ hat_X)
  beta[i] <- coef(m)[2]
}
mean(beta)
```

```
## [1] 1.998384
```

We now obtain the correct value on average. Using the confounder, we can obtain the value already with a sample size of 100:

```
reps <- 1000
beta <- numeric(reps)
for (i in 1:reps) {
  N <- 100
  C <- rnorm(N)
  Z <- rnorm(N)
  X <- 0.5 * Z + 0.5 * C + rnorm(N)
  Y <- 2 * X + C + rnorm(N)

  m <- lm(Y ~ X + C)
  beta[i] <- coef(m)[2]
}
mean(beta)
```

```
## [1] 2.002647
```

Exercise 6.2 Latin square

Problem

The effect of insulin on the blood concentration of glucose was studied on rabbits. Three rabbits received insulin doses A, B and C (corresponding to respectively 0, 1 and 2 units) on different days. The experiment is given below (dataset available in file `rabbit.txt`) with the glucose measurements (mg pr. 100 ml blood) taken 50 minutes after injection.

	Rabbit					
Day	1		2		3	
1	A	50	C	39	B	36
2	C	37	B	51	A	53
3	B	51	A	60	C	37

Make the *Table of Variables*, set up the associated statistical model, and analyse the data. Remember to obtain estimates of effects of interest.

Why is it not possible to investigate whether there is an interaction between rabbit and dose based on these data?

(Data are from Young & Romans (1948): Assay of insulin with one blood sample per rabbit per day. *Biometrics*, **4**, 122–131.)

Solution

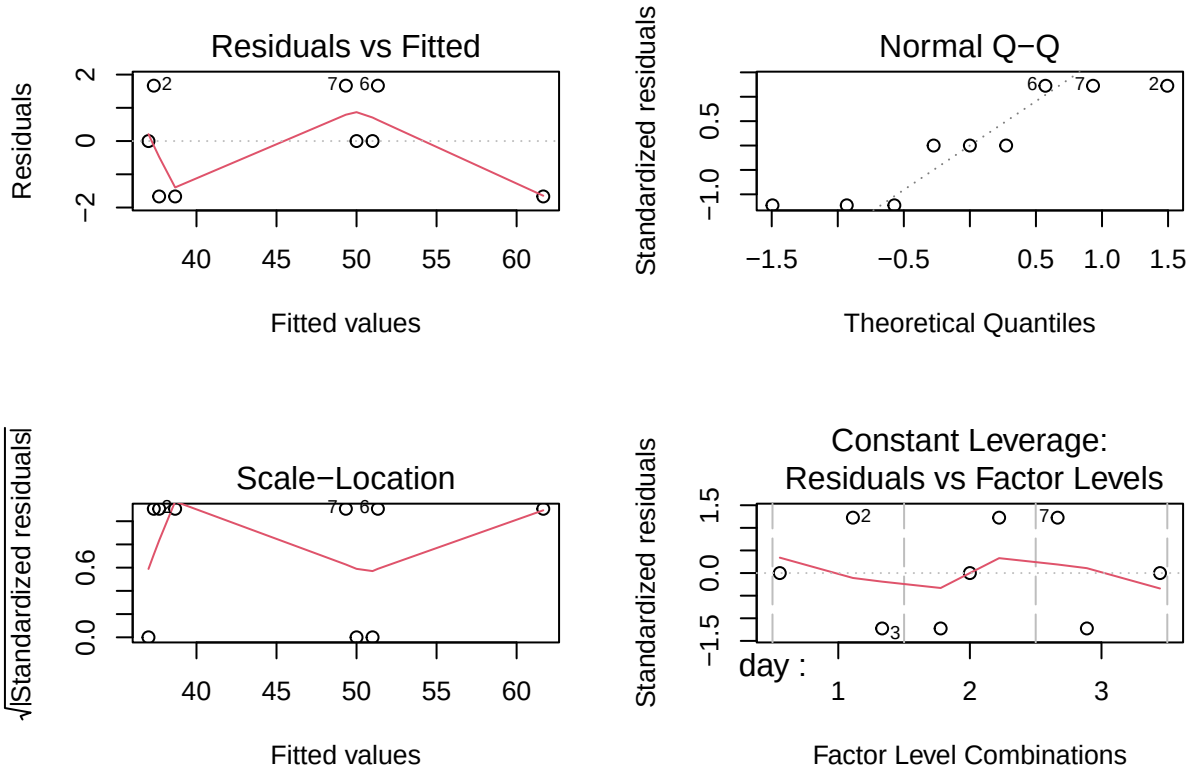
We load the data:

```
rabbit <- read.delim("rabbit.txt")
rabbit$day <- factor(rabbit$day)
rabbit$rabbit <- factor(rabbit$rabbit)
summary(rabbit)
```

```
##  day   rabbit     dose              glucose
##  1:3   1:3    Length:9           Min.   :36
##  2:3   2:3    Class :character   1st Qu.:37
##  3:3   3:3    Mode  :character   Median :50
##                                  Mean   :46
##                                  3rd Qu.:51
##                                  Max.   :60
```

We create and validate a model:

```
m <- lm(glucose ~ day + rabbit + dose, data = rabbit)
par(mfrow = c(2, 2))
plot(m)
```




```
par(mfrow = c(1, 1))
```

It is exceedingly difficult to tell whether this model is valid from the small number of data points but certainly normality looks borderline. We will continue anyway! Is there an effect of dose?

```
drop1(m, test = "F")
```

```
## Single term deletions
##
## Model:
## glucose ~ day + rabbit + dose
##        Df Sum of Sq    RSS    AIC F value  Pr(>F)
## <none>               16.67 19.546
## day     2     92.67 109.33 32.475    5.56 0.15244
## rabbit  2     96.00 112.67 32.745    5.76 0.14793
## dose    2    416.67 433.33 44.869   25.00 0.03846 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Yes, dose is significant ($p = 0.03846$). We can compare the average values within each dose using `emmeans`:

```
library(emmeans)
multcomp::cld(emmeans(m, ~dose), Letters = letters, details = TRUE)
```

```
## $emmeans
##  dose emmean   SE df lower.CL upper.CL .group
##  C      37.7 1.67  2     30.5     44.8  a
##  B      46.0 1.67  2     38.8     53.2  ab
##  A      54.3 1.67  2     47.2     61.5   b
##
## Results are averaged over the levels of: day, rabbit
## Confidence level used: 0.95
```

```
## P value adjustment: tukey method for comparing a family of 3 estimates
## significance level used: alpha = 0.05
## NOTE: If two or more means share the same grouping symbol,
##       then we cannot show them to be different.
##       But we also did not show them to be the same.
##
## $comparisons
##  contrast estimate   SE df t.ratio p.value
##  B - C        8.33 2.36  2   3.536  0.1274
##  A - C       16.67 2.36  2   7.071  0.0352
##  A - B        8.33 2.36  2   3.536  0.1274
##
## Results are averaged over the levels of: day, rabbit
## P value adjustment: tukey method for comparing a family of 3 estimates
```

We see that doses A and C are different but B could be similar to either A or C.

It is not possible to add interactions between the variables because we only observe each combination of rabbit and dose a single time. The resulting model has 11 parameters for the mean model (1 for the intercept, 2 to model the effect of day and 8 to model the interaction between dose and rabbit) which is greater than the number of observations so we cannot estimate the parameters.

Exercise 6.3 Cover crops for apples

Problem

In East Malling the total harvest of apples (in pounds) in a four year experimental period was investigated in a randomized block design with six treatments (cover crops A, ..., F) and four blocks. The design was implemented with 6 plots per block randomized over treatments. Beside the total harvest y in the experimental period, the total harvest x in a 4 years period prior to treatment was also recorded. The data are as follows (dataset available in file `apples.txt`):

	Block							
	1		2		3		4	
Cover crop	x	y	x	y	x	y	x	y
A	8.2	287	9.4	290	7.7	254	8.5	307
B	8.2	271	6.0	209	9.1	243	10.1	348
C	8.2	234	7.0	210	9.7	286	9.9	371
D	5.7	189	5.5	205	10.2	312	10.3	375
E	6.1	210	7.0	276	8.7	279	8.1	344
F	7.6	222	10.1	301	9.0	238	10.5	257

Analyse the data! Is there a significant effect of cover crop on the total harvest? Does the total harvest in the experimental period depend on the total harvest in the preceding period?

Assume that the previous harvest x was not measured. Is it then possible to find significant differences between the effect of cover crop?

Remark: The variables y and x appear to be recorded on different scales, i.e. the values of y approximately 30 times as big as the values of x. You can ignore this difference in scale, and simply use x as a covariate in the analysis of y.

Solution

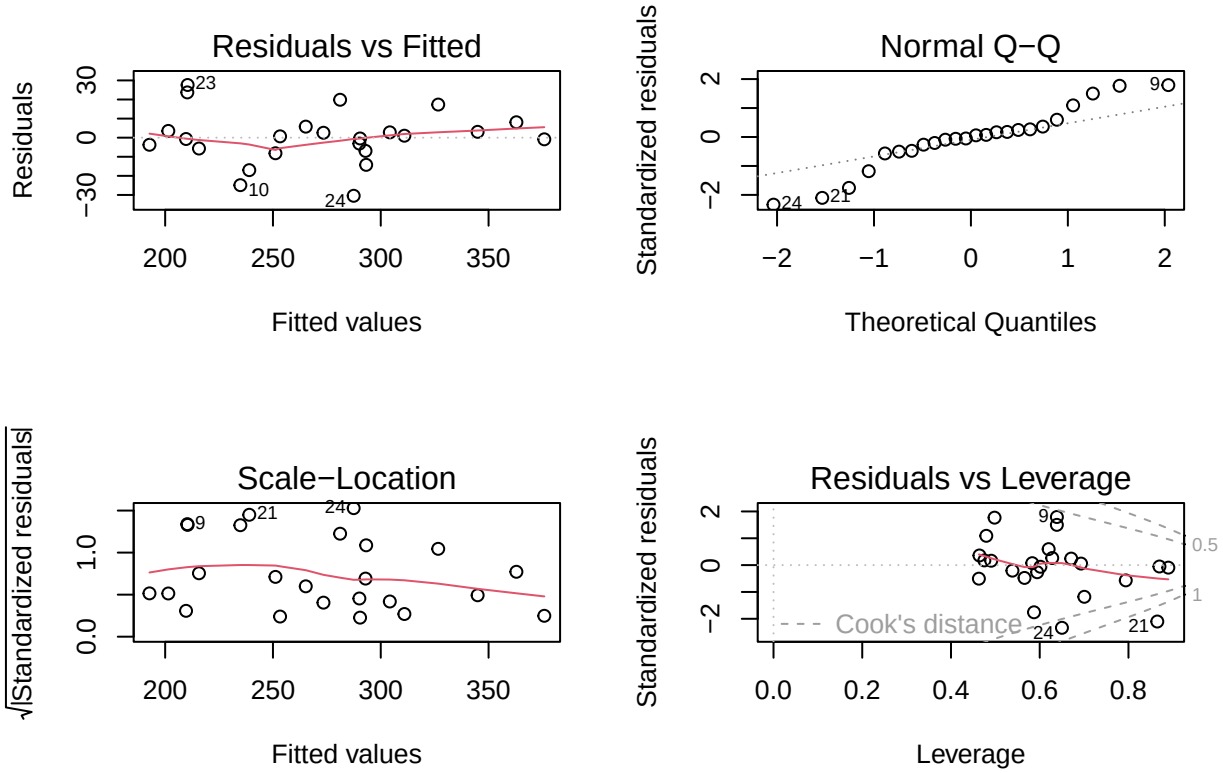
We load the data:

```
apple <- read.delim("apples.txt")
apple$block <- factor(apple$block)
summary(apple)
```

```
##     cover           block       x                y
##  Length:24          1:6   Min.   : 5.500   Min.   :189.0
##  Class :character   2:6   1st Qu.: 7.000   1st Qu.:231.0
##  Mode  :character   3:6   Median : 8.350   Median :273.5
##                     4:6   Mean   : 8.308   Mean   :271.6
##                           3rd Qu.: 9.750   3rd Qu.:302.5
##                           Max.   :10.500   Max.   :375.0
```

We fit an ANCOVA model and validate it:

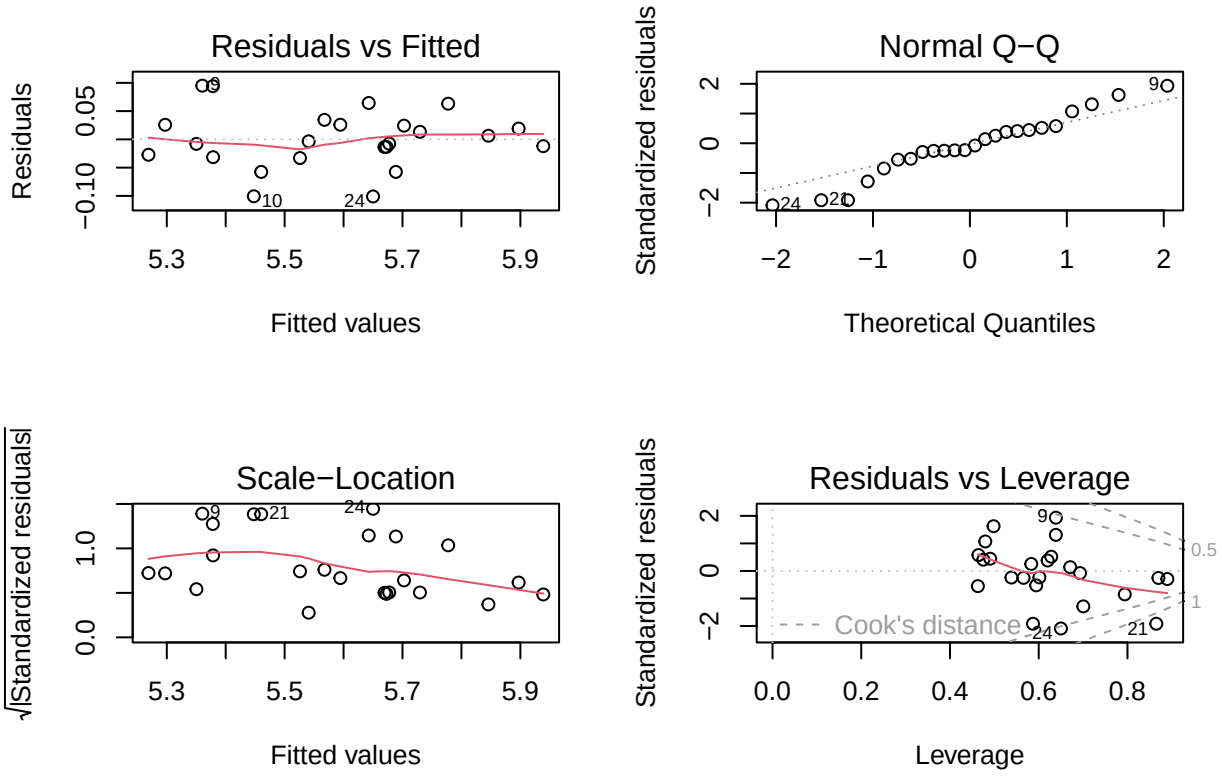
```
m1 <- lm(y ~ block + cover * x, data = apple)
par(mfrow = c(2, 2))
plot(m1)
```




```
par(mfrow = c(1, 1))
```

The residuals do not look normally distributed; some of them are too large. This usually indicates the need for a logarithmic transformation of the response. We try this and validate the model:

```
m2 <- lm(log(y) ~ block + cover * x, data = apple)
par(mfrow = c(2, 2))
plot(m2)
```




```
par(mfrow = c(1, 1))
```

This looks better! We try to reduce the model:

```
drop1(m2, test = "F")
```

```
## Single term deletions
##
## Model:
## log(y) ~ block + cover * x
##         Df Sum of Sq      RSS      AIC F value  Pr(>F)
## <none>               0.060112 -113.750
## block    3  0.118555 0.178667  -93.607  5.9167 0.01636 *
## cover:x  5  0.059425 0.119537 -107.252  1.7794 0.21318
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can drop the interaction between `cover` and `x`:

```
m3 <- lm(log(y) ~ block + cover + x, data = apple)
drop1(m3, test = "F")
```

```
## Single term deletions
##
## Model:
## log(y) ~ block + cover + x
##        Df Sum of Sq     RSS      AIC F value   Pr(>F)
## <none>              0.11954 -107.252
## block   3  0.074313 0.19385 -101.649  2.9012  0.07210 .
## cover   5  0.136614 0.25615  -98.961  3.2000  0.03917 *
## x       1  0.294130 0.41367  -79.458 34.4480 4.08e-05 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Although `block` is not significant, it would increase AIC significantly if we drop it, so we keep it. We conclude that there is a highly significant effect of cover on the total harvest ($p = 4cdot10^{-5}$). There is also a significant effect of the previous harvest on the current harvest ($p = 0.04$).

The model parameters an confidence intervals are:

```
cbind(estimate = coef(m3), confint(m3))
```

```
##                 estimate       2.5 %      97.5 %
## (Intercept)  4.716019758  4.40022427  5.03181524
## block2       0.006163623 -0.10938644  0.12171369
## block3      -0.079264485 -0.21840972  0.05988075
## block4       0.078820997 -0.07268536  0.23032735
## coverB      -0.064940039 -0.20513612  0.07525604
## coverC      -0.043713658 -0.18390974  0.09648242
## coverD      -0.031875487 -0.17359869  0.10984772
## coverE       0.068869920 -0.07666189  0.21440173
## coverF      -0.209247187 -0.35350308 -0.06499129
## x            0.110170000  0.06991080  0.15042920
```

We use `emmeans` to get an idea of the size of the effect of `cover`:

```
library(emmeans)
confint(emmeans(m3, ~cover))
```

```
##  cover emmean     SE df lower.CL upper.CL
##  A       5.63 0.0463 14     5.53     5.73
##  B       5.57 0.0462 14     5.47     5.67
##  C       5.59 0.0462 14     5.49     5.69
##  D       5.60 0.0468 14     5.50     5.70
##  E       5.70 0.0488 14     5.60     5.81
##  F       5.42 0.0498 14     5.32     5.53
##
## Results are averaged over the levels of: block
## Results are given on the log (not the response) scale.
## Confidence level used: 0.95
```

```
multcomp::cld(emmeans(m3, ~cover), Letters = letters)
```

```
##  cover emmean     SE df lower.CL upper.CL .group
##  F       5.42 0.0498 14     5.32     5.53  a
##  B       5.57 0.0462 14     5.47     5.67  ab
##  C       5.59 0.0462 14     5.49     5.69  ab
##  D       5.60 0.0468 14     5.50     5.70  ab
##  A       5.63 0.0463 14     5.53     5.73  ab
##  E       5.70 0.0488 14     5.60     5.81   b
##
## Results are averaged over the levels of: block
## Results are given on the log (not the response) scale.
## Confidence level used: 0.95
## P value adjustment: tukey method for comparing a family of 6 estimates
## significance level used: alpha = 0.05
## NOTE: If two or more means share the same grouping symbol,
##       then we cannot show them to be different.
##       But we also did not show them to be the same.
```

We see that only covers E and F are significantly different. To get interpretable estimates, we need to back-transform from the log scale. This results in confidence intervals for the ratios between covers:

```
confint(pairs(emmeans(m3, ~cover), type = "response"))
```

```
##  contrast ratio     SE df lower.CL upper.CL
##  A / B    1.067 0.0698 14    0.861     1.32
##  A / C    1.045 0.0683 14    0.843     1.29
##  A / D    1.032 0.0682 14    0.831     1.28
##  A / E    0.933 0.0633 14    0.747     1.17
##  A / F    1.233 0.0829 14    0.989     1.54
##  B / C    0.979 0.0640 14    0.790     1.21
##  B / D    0.967 0.0637 14    0.780     1.20
##  B / E    0.875 0.0589 14    0.701     1.09
##  B / F    1.155 0.0782 14    0.925     1.44
##  C / D    0.988 0.0650 14    0.796     1.23
##  C / E    0.894 0.0602 14    0.716     1.11
##  C / F    1.180 0.0799 14    0.945     1.47
##  D / E    0.904 0.0596 14    0.728     1.12
##  D / F    1.194 0.0839 14    0.948     1.50
##  E / F    1.321 0.0974 14    1.037     1.68
##
## Results are averaged over the levels of: block
## Confidence level used: 0.95
## Conf-level adjustment: tukey method for comparing a family of 6 estimates
## Intervals are back-transformed from the log scale
```

If we omit the previous harvest `x` from the model then we are no longer able to detect an effect of `cover`:

```
drop1(lm(log(y) ~ block + cover, data = apple), test = "F")
```

```
## Single term deletions
##
## Model:
## log(y) ~ block + cover
##        Df Sum of Sq     RSS     AIC F value  Pr(>F)
## <none>              0.41367 -79.458
## block   3   0.43133 0.84499 -68.316  5.2134 0.01151 *
## cover   5   0.03313 0.44680 -87.609  0.2403 0.93836
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This means that using the additional information about last years harvest results in more powerful analysis!

Exercise 6.4 Constructing a latin square design

Problem

The objective of this exercise is to use the AlgDesign-package in R to generate the latin square design from Exercise 6.2. The full factorial design with $3^3 = 27$ observations may be generated by the following code:

```
library(AlgDesign)
full.factorial <- gen.factorial(levels=3, nVars=3,
                                varNames=c("Rabbit", "Day", "Dose"))
full.factorial
```

In this design the levels of the 3 variables are the numbers -1, 0, 1. To make it easier for us to look at the variables we may recode the levels, which may be done using the following code:

```
full.factorial <- with(full.factorial,data.frame(
  Rabbit=factor(c("Rabbit 1", "Rabbit 2", "Rabbit 3")[2+Rabbit]),
  Day=factor(c("Day 1", "Day 2", "Day 3")[2+Day]),
  Dose=factor(c("Dose A", "Dose B", "Dose C")[2+Dose])))
```

Have a look at the full factorial design with the recoded variable levels. Now use the `optFederov()` function to make a design with 9 observations where you have interest on the main effects of the 3 variables. Do you find the design from Exercise 6.2?

Remarks:

1. The `optFederov()` looks for a D-optimal design. And in this particular situation the *latin square design* is the D-optimal design, so you would expect that `optFederov()` will find the latin square. However, `optFederov()` does a random search, so you may be unlucky that is does not find the true optimum. To improve on this you may increase the `nRepeats`-option, e.g. using `nRepeats=1000`.
2. You may need to rename the levels of *Dose* and *Rabbit* to have an exact match with the table in Exercise 6.2. Using different names for the levels does, of course, not change the basic properties of the design.

Solution

We generate the full factorial design:

```
library(AlgDesign)
full_factorial <- gen.factorial(
  levels = 3, nVars = 3,
  varNames = c("Rabbit", "Day", "Dose")
)
full_factorial
```

```
##    Rabbit Day Dose
## 1      -1  -1   -1
## 2       0  -1   -1
## 3       1  -1   -1
## 4      -1   0   -1
## 5       0   0   -1
## 6       1   0   -1
## 7      -1   1   -1
## 8       0   1   -1
## 9       1   1   -1
## 10     -1  -1    0
## 11      0  -1    0
## 12      1  -1    0
## 13     -1   0    0
## 14      0   0    0
## 15      1   0    0
## 16     -1   1    0
## 17      0   1    0
## 18      1   1    0
## 19     -1  -1    1
## 20      0  -1    1
## 21      1  -1    1
## 22     -1   0    1
## 23      0   0    1
## 24      1   0    1
## 25     -1   1    1
```

```
## 26      0   1    1
## 27      1   1    1
```

The default encoding uses the numbers -1, 0 and 1 for the levels. We can recode this to make it easier to see what is happening:

```
full_factorial <- with(full_factorial, data.frame(
  Rabbit = factor(
    c("Rabbit 1", "Rabbit 2", "Rabbit 3")[2 + as.numeric(Rabbit)]
  ),
  Day = factor(c("Day 1", "Day 2", "Day 3")[2 + as.numeric(Day)]),
  Dose = factor(c("Dose A", "Dose B", "Dose C")[2 + as.numeric(Dose)])
))
full_factorial
```

```
##      Rabbit   Day   Dose
## 1  Rabbit 1 Day 1 Dose A
## 2  Rabbit 2 Day 1 Dose A
## 3  Rabbit 3 Day 1 Dose A
## 4  Rabbit 1 Day 2 Dose A
## 5  Rabbit 2 Day 2 Dose A
## 6  Rabbit 3 Day 2 Dose A
## 7  Rabbit 1 Day 3 Dose A
## 8  Rabbit 2 Day 3 Dose A
## 9  Rabbit 3 Day 3 Dose A
## 10 Rabbit 1 Day 1 Dose B
## 11 Rabbit 2 Day 1 Dose B
## 12 Rabbit 3 Day 1 Dose B
## 13 Rabbit 1 Day 2 Dose B
## 14 Rabbit 2 Day 2 Dose B
## 15 Rabbit 3 Day 2 Dose B
## 16 Rabbit 1 Day 3 Dose B
## 17 Rabbit 2 Day 3 Dose B
## 18 Rabbit 3 Day 3 Dose B
## 19 Rabbit 1 Day 1 Dose C
## 20 Rabbit 2 Day 1 Dose C
## 21 Rabbit 3 Day 1 Dose C
## 22 Rabbit 1 Day 2 Dose C
## 23 Rabbit 2 Day 2 Dose C
## 24 Rabbit 3 Day 2 Dose C
## 25 Rabbit 1 Day 3 Dose C
## 26 Rabbit 2 Day 3 Dose C
## 27 Rabbit 3 Day 3 Dose C
```

We use the `optFederov` function to find the latin square design:

```
latin_square <- optFederov(~ Rabbit + Day + Dose, full_factorial,
  nTrials = 9
)$design
latin_square
```

```
##      Rabbit   Day   Dose
## 3  Rabbit 3 Day 1 Dose A
## 4  Rabbit 1 Day 2 Dose A
## 8  Rabbit 2 Day 3 Dose A
## 10 Rabbit 1 Day 1 Dose B
## 14 Rabbit 2 Day 2 Dose B
```

```
## 18 Rabbit 3 Day 3 Dose B
## 20 Rabbit 2 Day 1 Dose C
## 24 Rabbit 3 Day 2 Dose C
## 25 Rabbit 1 Day 3 Dose C
```

We can use the `xtabs` function to more easily summarize the design:

```
xtabs(~ Rabbit + Day + Dose, latin_square)
```

```
## , , Dose = Dose A
##
##           Day
## Rabbit     Day 1 Day 2 Day 3
##   Rabbit 1     0     1     0
##   Rabbit 2     0     0     1
##   Rabbit 3     1     0     0
##
## , , Dose = Dose B
##
##           Day
## Rabbit     Day 1 Day 2 Day 3
##   Rabbit 1     1     0     0
##   Rabbit 2     0     1     0
##   Rabbit 3     0     0     1
##
## , , Dose = Dose C
##
##           Day
## Rabbit     Day 1 Day 2 Day 3
##   Rabbit 1     0     0     1
##   Rabbit 2     1     0     0
##   Rabbit 3     0     1     0
```

This is indeed the latin square!

Exercise 6.5 Non-linear regression

Problem

The following experiment was carried out in a greenhouse: 15 pots were sown with barley seeds: 3, 7, 15, 34, 77 barley seeds per pot, respectively, with three pots for each number of barley seeds. After harvest, the total fresh weight yields (in grams) were measured for each pot. The results are listed in the table below:

No. of seeds	Yield		
3	7.5	9.8	9.0
7	18.8	27.7	27.1
15	64.7	30.2	37.0
34	84.3	110.0	71.2
77	125.8	85.7	91.9

We want to use the following non-linear model for the relationship between number of barley seeds, $x =$ `seeds`, and the logarithmic yield, $y = \log($`yield`$)$:

$$y \approx a - b \cdot \mathrm{e}^{-cx} \tag{1}$$

Please answer the following questions:

1. Plot the logarithmic yield (variable y) against `seeds`.

2. What is the interpretation of the parameters a and b? Make a qualified guess on the values for a and b.

   Hint: What happens for $x = 0$ and x very large?

3. Although more difficult it is also possible to make a qualified guess on the c parameter, namely:

$$c \approx \frac{\log(2)}{15} \approx 0.045$$

   You are welcome to explain the reasoning behind this guess (if you can), but otherwise you may simply take it for granted.

4. Assuming that that data frame with the observations is called `barley`, the following R code makes an interactive plot in RStudio (remember to install the `manipulate`-package if you have not done it before):

```
library(manipulate)
manipulate(
  {plot(log(yield) ~ seeds, data=barley)
   x <- 0:80
   y <- a - b * exp(-c * x)
   lines(x, y)},
  a=slider(2, 8, initial=5, step=0.1),
  b=slider(0, 4, initial=2, step=0.1),
  c=slider(0, 0.1, initial=0.045, step=0.005)
)
```

   Try this and click on the *gear sprocket* (in danish: *tandhjul*) icon in the graphical window to manipulate the a, b and c parameters interactively. See if you can change the parameters such that the data points are fitted closely by the non-linear curve.

   Remark: In this way the `manipulate()` function may be used to derive initial guesses for the parameters in a non-linear regression. However, if you already have an adequate guess, then this is not necessary.

5. Use `nls()` with the initial guesses given in the `start`-option to fit the parameters a, b and c by a non-linear regression.

6. Give estimates and confidence intervals for the parameters a, b and c.

7. Is the non-linear model valid?

   Hint: You may make a *residual plot* and *normal quantile plot* by "hand" using the code on slide 9 (Day 6).

8. For some of the "classical" and often used non-linear functions there exist so-called *self-starting* functions in R. The non-linear function used in this exercise is one of these functions. The associated R function is called `SSasymp()`. Try the following R code and relate it to the results you found above:

```
m2 <- nls(log(yield) ~ SSasymp(seeds, a, a.minus.b, log.c), data=barley)
cbind(estimate=coef(m2), confint(m2))
```

   Remark: `SSasymp()` uses a different parametrization than the one used in Eq. (1). Can you describe how you pass between these parametrizations?

(Reference: Based on exercise 8.6 from Anders Tolver & Helle S{ø}rensen: *Lecture notes for Applied Statistics.*)

Solution

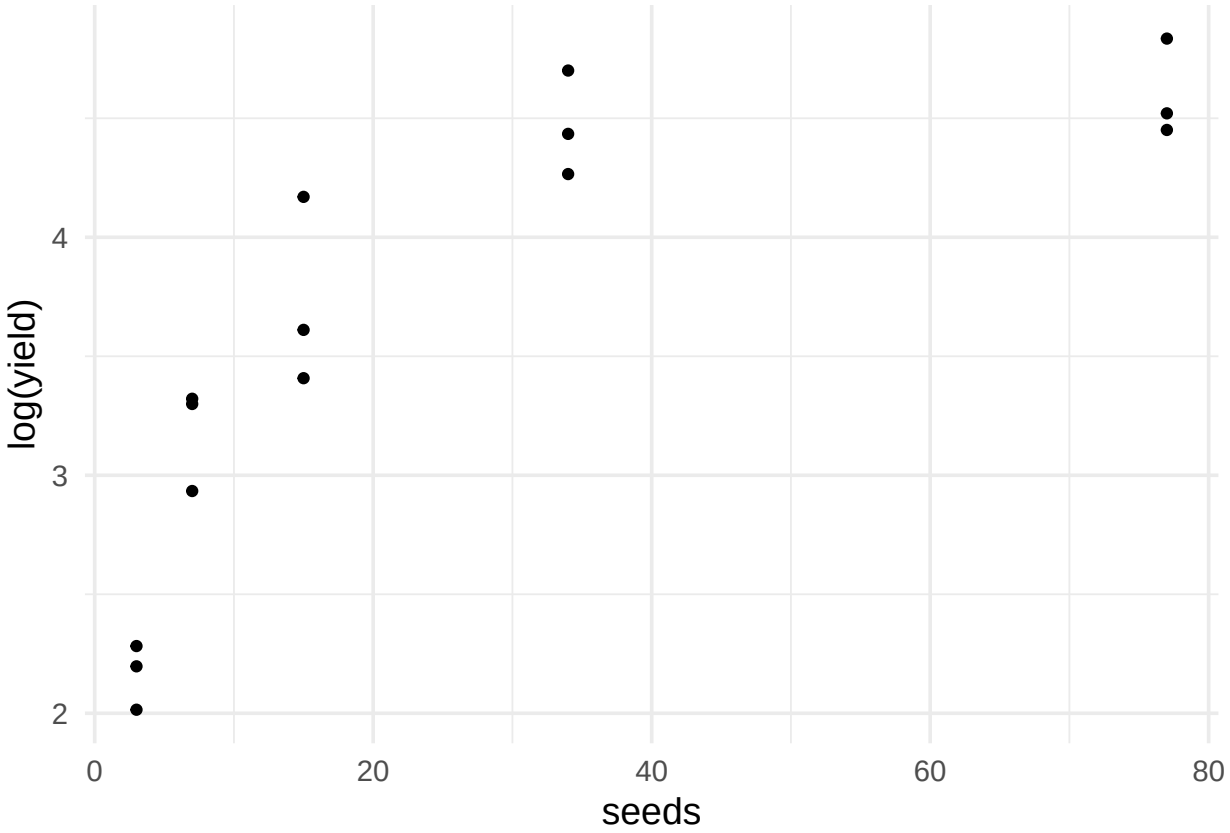
We read the data:

```
barley <- read.delim("barley.txt")
summary(barley)
```

```
##      seeds          yield
##  Min.   : 3.0   Min.   :  7.50
##  1st Qu.: 7.0   1st Qu.: 22.95
##  Median :15.0   Median : 37.00
##  Mean   :27.2   Mean   : 53.38
##  3rd Qu.:34.0   3rd Qu.: 85.00
##  Max.   :77.0   Max.   :125.80
```

We plot the variables:

```
library(ggplot2)
ggplot(barley, aes(x = seeds, y = log(yield))) +
  geom_point()
```



The parameter a in the model corresponds to the value of the logarithimic yield with "infinite seeds". The parameter b in the model is such that $a - b$ is the logarithmic yield with 0 seeds. From looking at the plot, a reasonable guess for a is around 4.5 while $a - b$ should be around 2 (so b is around 2.5).

As we are dealing with exponential growth we can use the rule of thumb that growth rate is log(2) divided by the doubling time. In this case, the doubling time is around 15 (from the plot) so we get an estimate for c of $\log(2)/15 \approx 0.045$.

It is possible to use the code below to find even better guesses for the parameters: `{r. eval=FALSE} library(manipulate) manipulate( {plot(log(yield) ~ seeds, data=barley) x <- 0:80 y <- a - b * exp(-c * x) lines(x, y)}, a=slider(2, 8, initial=5, step=0.1), b=slider(0, 4, initial=2, step=0.1), c=slider(0, 0.1, initial=0.045, step=0.005) )` Using the guesses from the code above, we can now fit the parameters by a non-linear regression:

```
m1 <- nls(log(yield) ~ a - b * exp(-c * seeds),
  start = list(a = 4.8, b = 2.6, c = 0.06), data = barley
)
```

We get estimates:

```
cbind(estimate = coef(m1), confint(m1))
```

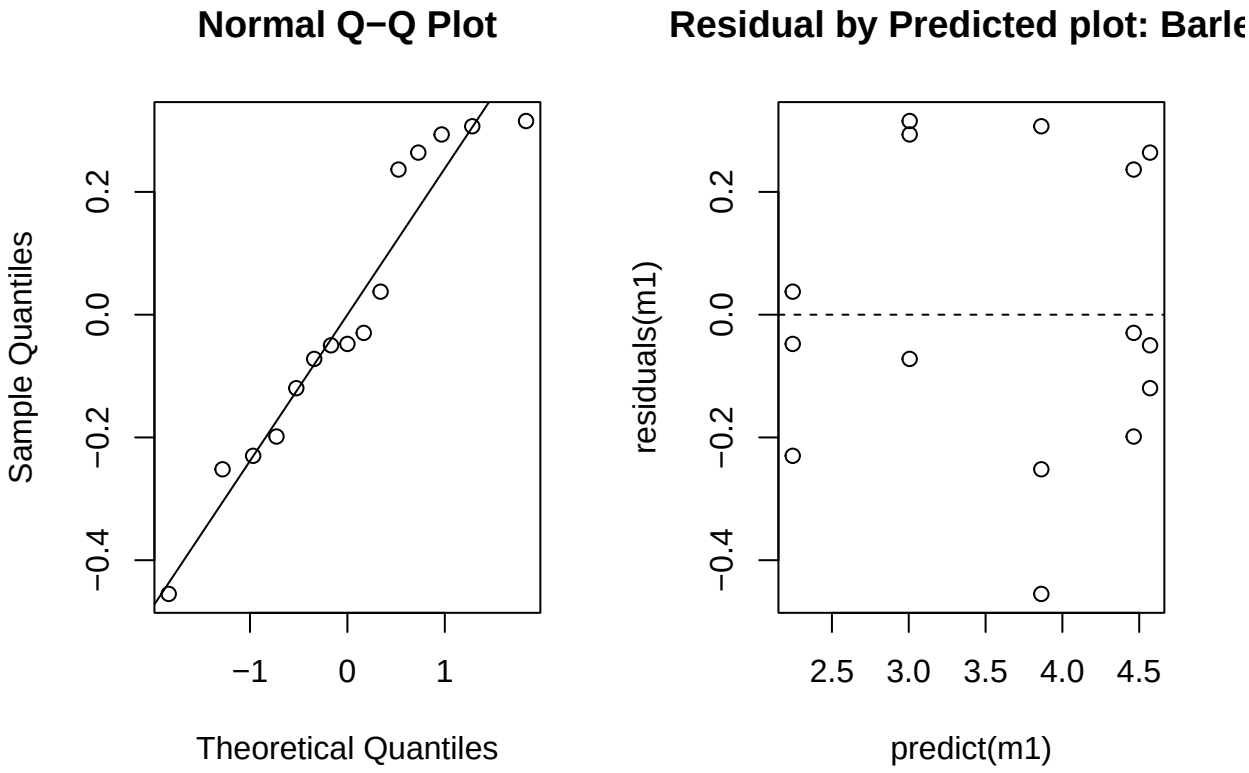
```
## Waiting for profiling to be done...
```

```
##     estimate       2.5%     97.5%
## a 4.57213011 4.30690162 4.8714858
## b 3.13216777 2.53800764 3.8987141
## c 0.09900851 0.06094905 0.1597193
```

None of the confidence intervals are close to 0 so we will omit significance testing (as all parameters are surely significant). We validate the model:

```
par(mfrow = c(1, 2))
qqnorm(residuals(m1))
abline(mean(residuals(m1)), sd(residuals(m1)))

plot(predict(m1), residuals(m1), main = "Residual by Predicted plot: Barley")
abline(0, 0, lty = 2)
```




```
par(mfrow = c(1, 1))
```

The plots look good! We try the self-starting parameterization:

```
m2 <- nls(log(yield) ~ SSasymp(seeds, a, a.minus.b, log.c), data = barley)
cbind(estimate = coef(m2), confint(m2))
```

```
## Waiting for profiling to be done...
```

```
##            estimate       2.5%     97.5%
```

```
## a          4.572127  4.3069016  4.871486
## a.minus.b  1.439952  0.5850706  2.049297
## log.c     -2.312541 -2.7980108 -1.834396
```

We can use the naming of the variables to convert the results here to the other parametrization; the first parameter is the same in both `m1` and `m2`. To recover b from `m2` we compute the difference between the first and second parameters (the `unname` function removes an incorrect name applied to the result):

```
unname(coef(m2)[1] - coef(m2)[2])
```

```
## [1] 3.132175
```

To obtain c, we take the exponential of the third parameter:

```
unname(exp(coef(m2)[3]))
```

```
## [1] 0.09900937
```

These results agree with what we found using `m1`.