CHAPTER

11

General Linear Model (ANCOVA)

OUTLINE

11.1 INTRODUCTION

The "model of the mean," t-test, simple linear regression, and analysis of variance (ANOVA) are all just special cases of a very general and powerful statistical model, the general linear model (≠ GLM!; see Chapter 13). This model expresses a continuous response as a linear combination of the effects of discrete and/or continuous explanatory variables plus a single random contribution from a normal distribution, whose variance is estimated along with the coefficients of all discrete and continuous covariates and possible interactions.

Before this was widely recognized, people used to make a rather sharp and artificial distinction between linear models that contain categorical explanatory variables only and were called t-test or ANOVA models and those that contain continuous covariates only and were called regression models. Models that contained both types of explanatory variables were usually treated as ANOVAs with typically a single continuous covariate to correct for preexisting variation among experimental units. These

FIGURE 11.1 Male asp viper (*Vipera aspis*), Switzerland, 2007. (*Photo T. Ott*)

models were called analysis of covariance (ANCOVA) models and that's why I use this term here.

Nowadays, in many practical applications, we typically have several explanatory variables of both types. In addition, we want to fit both main effects of these covariates and some or all their pairwise or even higher-order interactions. As an example, we here consider an ANCOVA model with an interaction between a discrete and a continuous covariate. We saw how to fit interactions between two discrete covariates in Chapter 10. Interactions between two continuous covariates are easy to fit: simply fit one additional covariate whose values are obtained by multiplication of the two main covariates.

In this chapter, we consider the relationship between body mass and body length of the asp viper (Fig. 11.1) in three populations: Pyrenees, Massif Central, and the Jura mountains. We are particularly interested in population-specific differences of the mass–length relationship, i.e., in the interaction between length and population. The means parameterization of the model we will fit can be written as (see Section 6.3.6)

$$y_i = \alpha_{j(i)} + \beta_{j(i)} * x_i + \varepsilon_i$$

and

$$\varepsilon_i \sim \text{Normal}(0, \sigma^2),$$

where y_i is the body mass of individual i, $\alpha_{j(i)}$ and $\beta_{j(i)}$ are the intercept and the slope, respectively, of the mass–length relationship in population j, x_i is the body length of snake i, and as usual, ε_i describes the combined effects of all unmeasured influences on the body mass of snake i and is assumed to behave like a normal random variable whose variance σ^2 we estimate.

The effects parameterization of the same model is this:

$$y_i = \alpha_{\text{Pyr}} + \beta_1 * x_{\text{MC}(i)} + \beta_2 * x_{\text{Jura}(i)} + \beta_3 * x_{\text{body}(i)} + \beta_4 * x_{\text{body}(i)} * x_{\text{MC}(i)} \\ + \beta_5 * x_{\text{body}(i)} * x_{\text{Jura}(i)} + \varepsilon_i$$

In addition to y_i and ε_i that are as before, α_{Pyr} is the expected mass of snakes in the Pyrenees, β_1 is the difference between the expected mass of snakes in the Massif Central and that in the Pyrenees, and $x_{\text{MC}(i)}$ is the indicator for snakes caught in the Massif Central. β_2 is the difference between the expected mass in the Jura and that in the Pyrenees, $x_{\text{Jura}(i)}$ is the indicator for snakes in the Jura, β_3 is the slope of the regression of body mass on body length x_{body} in the Pyrenees, β_4 is the difference in that slope between the Massif Central and the Pyrenees, and β_5 is the difference of slopes between the Jura and the Pyrenees. Thus, snakes in the Pyrenees act as baseline with which snakes from the Massif Central and the Jura are compared, but as usual, this choice has no effect on inference.

11.2 DATA GENERATION

As always, we assume a balanced design simply for convenience of data generation.

```
n.groups <- 3
n.sample <- 10
n <- n.groups * n.sample # Total number of data points
x <- rep(1:n.groups, rep(n.sample, n.groups)) # Indicator for
population
pop <- factor(x, labels = c("Pyrenees", "Massif Central", "Jura"))
length <- runif(n, 45, 70)     # Obs. body length (cm) is rarely less
than 45
```

We build the design matrix of an interactive combination of length and population, inspect that, and select the parameter values, i.e., choose values for α_{Pyr}, β_1, β_2, β_3, β_4, and β_5.

```
Xmat <- model.matrix(~ pop*length)
print(Xmat, dig = 2)
beta.vec <- c(-250, 150, 200, 6, -3, -4)
```

Next, we build up the body mass measurements y_i by adding the residual to the value of the linear predictor, with residuals drawn from an appropriate zero-mean normal distribution. The value of the linear predictor is obtained by matrix multiplication of the design matrix (`Xmat`) and the parameter vector (`beta.vec`). Our vipers are probably all too fat, but that doesn't really matter for our purposes.

```
lin.pred <- Xmat[,] %*% beta.vec          # Value of lin.predictor
eps <- rnorm(n = n, mean = 0, sd = 10)    # residuals
mass <- lin.pred + eps                    # response = lin.pred + residual
hist(mass)                                # Inspect what we've created
matplot(cbind(length[1:10], length[11:20], length[21:30]), cbind(mass[1:10],
mass[11:20], mass[21:30]), ylim = c(0, max(mass)), ylab = "Body mass (g)", xlab =
"Body length (cm)", col = c("Red","Green","Blue"), pch = c("P","M","J"), las = 1,
cex = 1.2, cex.lab = 1.5)
```

We have created a data set in which vipers from the Pyrenees have the steepest slope between mass and length, followed by those from the Massif Central and finally those in the Jura mountains (Fig. 11.2). Now let's disassemble these data.

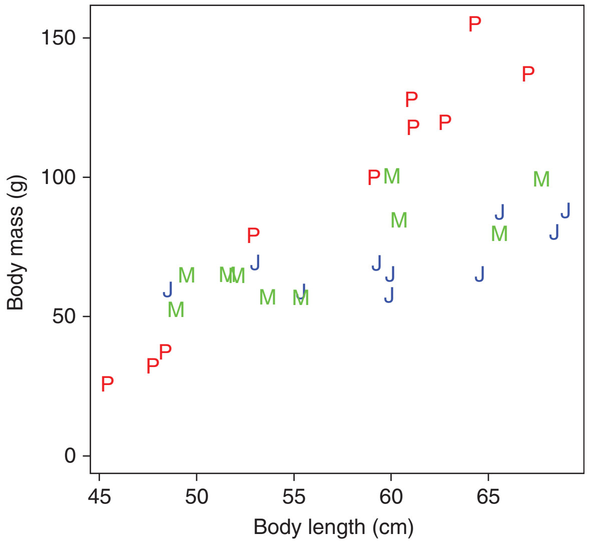

FIGURE 11.2 Simulated data set showing body mass versus length of 10 asp vipers in each of three populations (P – Pyrenees, M – Massif Central, and J – Jura).

11.3 ANALYSIS USING R

The code for an analysis in R is very parsimonious indeed:

```
summary(lm(mass ~ pop * length))
> summary(lm(mass ~ pop * length))
[...]
Coefficients:

                          Estimate  Std. Error  t value   Pr(>|t|)
(Intercept)              -246.6623    24.8588   -9.923   5.72e-10 ***
popMassif Central         197.5543    37.8767    5.216   2.41e-05 ***
popJura                   241.5563    39.1405    6.172   2.24e-06 ***
length                      5.9623     0.4323   13.792   6.66e-13 ***
popMassif Central:length   -3.8041     0.6631   -5.737   6.52e-06 ***
popJura:length             -4.7114     0.6595   -7.144   2.20e-07 ***
[...]
Residual standard error: 10.01 on 24 degrees of freedom
Multiple R-squared: 0.9114,     Adjusted R-squared: 0.8929
F-statistic: 49.37 on 5 and 24 DF, p-value: 7.48e-12
```

These coefficients can directly be compared with the beta vector since we simulated the data exactly in the default effects format of a linear model specified in R. The residual standard deviation is called residual standard error by R.

```
beta.vec
cat("And the residual SD was 10 \n")
```

```
> beta.vec
[1] -250  150  200     6   -3   -4
> cat("And the residual SD was 10 \n")
And the residual SD was 10
```

11.4 ANALYSIS USING WinBUGS (AND A CAUTIONARY TALE ABOUT THE IMPORTANCE OF COVARIATE STANDARDIZATION)

In WinBUGS, I find it much easier to fit the means parameterization of the model, i.e., to specify three separate linear regressions for each mountain range. The effects (i.e., differences of intercept or slopes with reference to the Pyrenees) are trivially easy to recover as derived parameters by just adding a few WinBUGS code lines. This allows for better comparison between input and output values.

```
# Define model
sink("lm.txt")
cat("
model {

# Priors
 for (i in 1:n.group){
    alpha[i] ~ dnorm(0, 0.001)      # Intercepts
    beta[i] ~ dnorm(0, 0.001)       # Slopes
 }
 sigma ~ dunif(0, 100)        # Residual standard deviation
 tau <- 1 / ( sigma * sigma)

# Likelihood
 for (i in 1:n) {
   mass[i] ~ dnorm(mu[i], tau)
   mu[i] <- alpha[pop[i]] + beta[pop[i]]* length[i]
 }

# Derived quantities
# Define effects relative to baseline level
 a.effe2 <- alpha[2] - alpha[1]      # Intercept Massif Central vs. Pyr.
 a.effe3 <- alpha[3] - alpha[1]      # Intercept Jura vs. Pyr.
 b.effe2 <- beta[2] - beta[1]        # Slope Massif Central vs. Pyr.
 b.effe3 <- beta[3] - beta[1]        # Slope Jura vs. Pyr.

# Custom tests
 test1 <- beta[3] - beta[2]          # Slope Jura vs. Massif Central
}
",fill=TRUE)
sink()

# Bundle data
win.data <- list(mass = as.numeric(mass), pop = as.numeric(pop), length = length,
n.group = max(as.numeric(pop)), n = n)

# Inits function
inits <- function(){ list(alpha = rnorm(n.group, 0, 2), beta = rnorm(n.groups, 1,
1), sigma = rlnorm(1))}

# Parameters to estimate
parameters <- c("alpha", "beta", "sigma", "a.effe2", "a.effe3", "b.effe2",
"b.effe3", "test1")

# MCMC settings
ni <- 1200
nb <- 200
nt <- 2
nc <- 3
```

```
# Start Markov chains
out <- bugs(win.data, inits, parameters, "lm.txt", n.thin=nt, n.chains=nc,
n.burnin=nb, n.iter=ni, debug = TRUE)
```

This is a simple model that converges rapidly. We inspect the results and compare them with the "truth" in the data-generating random process as well as with the inference from R …

```
print(out, dig = 3)                # Bayesian analysis
beta.vec                           # Truth in the data-generating process
summary(lm(mass ~ pop * length))   # The ML solution again
```

… and are perplexed! WinBUGS claims that the Markov chains have converged (see Rhat values), but we get totally different estimates from what we should! Remember that `alpha[1]` and `beta[1]` in WinBUGS correspond to the intercept and the `length` main effect in the analysis in R and `a.effe2`, `a.effe3`. `b.effe2`, `b.effe3` to the remaining terms of the analysis in R.

```
> print(out, dig = 3)# Bayesian analysis
Inference for Bugs model at "lm.txt", fit using WinBUGS,
 3 chains, each with 1200 iterations (first 200 discarded), n.thin = 2
 n.sims = 1500 iterations saved
             mean     sd     2.5%      25%      50%     75%    97.5%  Rhat n.eff
alpha[1] -100.524 31.918 -162.910 -122.400 -100.600 -79.105  -38.406 1.002  1500
alpha[2]  -16.420 26.238  -65.813  -34.060  -16.670   1.595   34.926 1.000  1500
alpha[3]   -2.613 26.127  -51.847  -20.862   -2.920  15.893   47.693 1.001  1500
beta[1]     3.442  0.557    2.336    3.066    3.443   3.813    4.540 1.002  1500
beta[2]     1.586  0.464    0.669    1.272    1.588   1.896    2.467 1.000  1500
beta[3]     1.205  0.438    0.367    0.885    1.207   1.503    2.059 1.002  1500
sigma      15.841  3.089   10.669   13.650   15.525  17.637   22.610 1.001  1500
a.effe2    84.103 38.764    6.313   57.752   84.620 110.300  160.082 1.002  1500
a.effe3    97.911 41.127   16.015   70.735   97.855 126.200  175.105 1.001  1500
b.effe2    -1.856  0.684   -3.178   -2.327   -1.856  -1.401   -0.479 1.002  1500
b.effe3    -2.237  0.708   -3.574   -2.716   -2.253  -1.777   -0.820 1.001  1500
test1      -0.381  0.633   -1.608   -0.782   -0.396   0.032    0.866 1.001  1500
deviance  249.071  8.258  232.847  243.300  249.000 254.800  264.552 1.000  1500
[ ... ]

> beta.vec                         # Truth in the data-generating process
[1]  -250  150  200   6   -3   -4
```

So `lm()` is able to recover the right parameter values, up to sampling and estimation error, but WinBUGS is not. Why is this?

The problem turns out to reside in the lack of standardization of the covariate `length`. In WinBUGS, it is always advantageous to scale covariates so that their extremes are not too far away from zero; otherwise, there may be nonconvergence or other problems. And the ugly thing here is that from looking at the convergence diagnostics (Rhat), we would never have guessed that there was a problem!

This example illustrates how useful it is to check the consistency of one's inference from WinBUGS with other sources, e.g., estimates from a simpler, but similar model run in WinBUGS or maximum likelihood estimates from another software. Know thy model! Alternatively, we could also have plotted the estimated regression lines into the observed data and would have seen easily that something was wrong.

As a quick check that lack of standardization was indeed the problem, we repeat both the maximum likelihood and the Bayesian analysis using a normalized version of the length covariate. This is simple; we just need to redefine the `length` covariate in the data list we pass to WinBUGS and can rerun the same code as before.

```
# Data passed to WinBUGS
win.data <- list(mass = as.numeric(mass), pop = as.numeric(pop), length =
as.numeric(scale(length)), n.group = max(as.numeric(pop)), n = n)

# Start Markov chains
out <- bugs(win.data, inits, parameters, "lm.txt", n.thin=nt, n.chains=nc,
n.burnin=nb, n.iter=ni, debug = FALSE)

...

# Inspect results
print(out, dig = 3)
> print(out, dig = 3)
[ ... ]
            mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
alpha[1]  97.796 3.406  90.679  95.820  97.905 100.100 104.252 1.004  1500
alpha[2]  74.952 3.504  67.864  72.690  75.030  77.292  81.580 1.004  1200
alpha[3]  66.535 3.653  59.009  64.060  66.660  68.995  73.265 1.007   350
beta[1]   41.243 3.158  34.829  39.150  41.215  43.332  47.316 1.005   870
beta[2]   14.486 3.763   7.318  11.947  14.570  16.920  22.112 1.002  1500
beta[3]    8.868 3.820   1.635   6.401   8.803  11.322  16.816 1.000  1500
sigma     10.587 1.678   7.937   9.380  10.370  11.510  14.516 1.005   630
a.effe2  -22.844 4.756 -31.771 -25.962 -22.815 -19.718 -13.498 1.003  1200
a.effe3  -31.262 5.000 -41.826 -34.470 -31.260 -28.050 -21.384 1.005   400
b.effe2  -26.757 4.802 -36.065 -30.032 -26.800 -23.620 -16.660 1.002  1500
b.effe3  -32.375 4.907 -41.445 -35.853 -32.350 -29.345 -22.162 1.001  1500
test1     -5.618 5.334 -16.024  -9.007  -5.836  -2.101   5.108 1.001  1500
deviance 225.218 4.558 218.700 221.900 224.500 227.800 235.500 1.009   270
[ ... ]
```

Compare with MLEs (R output slightly edited):

```
print(lm(mass ~ pop * as.numeric(scale(length)))$coefficients, dig = 4)
...
> print(lm(mass ~ pop * as.numeric(scale(length)))$coefficients, dig = 4)
                (Intercept)       popMassif Central
                 98.94           -22.95
                 popJura           as.numeric(scale(length))
                -31.54             41.78
popMassif Central:as.numeric(scale(length))  popJura:as.numeric(scale(length))
                -26.66           -33.01
```

Indeed, we now get consistent estimates in both analyses. Hence, previously the scale of the covariate didn't allow WinBUGS to converge, even though the Rhat values reported did indicate convergence. As a cautionary principle, we might therefore always consider to transform all covariates for WinBUGS, even if that slightly complicates presentation of results afterwards (for instance, in graphics). Transforming can mean centering, i.e., subtracting the mean, which changes the intercept only, but not the slope. Transforming can also mean normalizing, i.e., subtracting the mean and dividing the result by the standard deviation of the original covariate values. This changes both the intercept and the slope relative to an analysis with the original covariate. In the above case, centering will also work (want to try this out?).

11.5 SUMMARY

In a key chapter for your understanding of the modeling of grouped data, we have looked at the general linear model, or ANCOVA, in R and WinBUGS. We have focused on a model with one discrete and one continuous predictor and their interaction. Understanding ANCOVA is an important intermediate step to understanding the linear-mixed model in the next chapter.

EXERCISES

1. *Probability of a parameter*: What is the probability that the slope of the mass–length relationship of asp vipers is inferior in the Jura than in the Massif Central? Produce a graphical and a numerical answer.
2. *Related models*: Adapt the code to fit two variations of the model:
   - Fit different intercepts but a common slope
   - Fit the same intercept and the same slope

3. *Quadratic effects*: Add a quadratic term to the mass–length relationship, i.e., fit the model `pop + (length + length^2)`. You do not need to reassemble a data set that contains an effect of length squared, but you can simply take the data set we have already created in this chapter.
4. *Swiss hare data*: Fit an ANCOVA (`pop * year`, with year as a continuous explanatory variable) to the mean density. Also compute residuals and plot them to check for outliers.