APPENDIX C

Software

C.1 Getting started with R, Bugs, and a text editor

Follow the instructions at `www.stat.columbia.edu/~gelman/arm/software/` to download, install, and set up R and Bugs on your Windows computer. The webpage is occasionally updated as the software improves, so we recommend checking back occasionally. R, OpenBugs, and WinBugs have online help with more information available at `www.r-project.org`, `www.math.helsinki.fi/openbugs/`, and `www.mrc-bsu.cam.ac.uk/bugs/`.

Set up a *working directory* on your computer for your R work. Every time you enter R, your working directory will automatically be set, and the necessary functions will be loaded in.

Configuring your computer display for efficient data analysis

We recommend working with three nonoverlapping open windows, as pictured in Figure C.1: an R console, the R graphics window, and a text editor (ideally a program such as Emacs or WinEdt that allows split windows, or the script window in the Windows version of R). When programming in Bugs, the text editor will have two windows open: a file (for example, `project.R`) with R commands, and a file (for example, `project.bug`) with the Bugs model. It is simplest to type commands into the text file with R commands and then cut and paste them into the R console. This is preferable to typing in the R console directly because copying and altering the commands is easier in the text editor. To run Bugs, there is no need to open a Bugs window; R will do this automatically when the function `bugs()` is called (assuming you have set up your computer as just described, which includes loading the `R2WinBUGS` package in R). The only reason to manually open a Bugs window is to access the manuals and examples in its Help menu.

Software updates

Here we discuss how to set up and run the statistical packages that we use to fit regressions and multilevel models. All this software is under development, so some of the details of the code and computer output in the book may change along with the programs. We recommend periodically checking the websites for R, Bugs, and other software and updating as necessary.

C.2 Fitting classical and multilevel regressions in R

Using R for classical regression and miscellaneous statistical operations

The `lm()` and `glm()` functions fit linear and generalized linear models in R. Many examples appear in Part 1 of this book; you can see the R documentation and other references given at the end of this chapter for instructions and further examples.

We have prepared several functions including `display()`, `sim()`, `se.coef()`, and

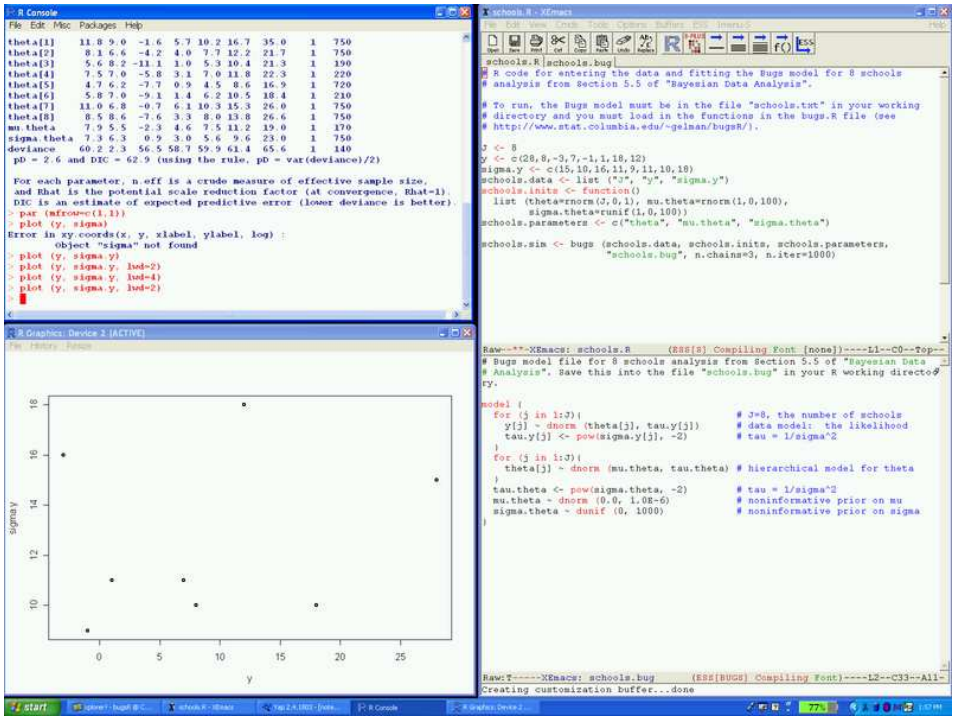

Figure C.1 *Configuration of a computer screen with R console, R graphics window, and a text editor (in this case, Xemacs) with two windows, one for R script and one for a Bugs model. We call Bugs from R, so there is no need to have an open Bugs window on the screen.*

`sigma.hat()`, for displaying, accessing, and generating simulations summarizing the inferences from linear and generalized linear models in R, as well as functions such as `bayesglm()` and `bayespolr()` for fitting Bayesian generalized linear models and ordered logistic regressions. All these functions, and a few others, are in the R package `arm` (*a*pplied *r*egression and *m*ultilevel modeling) and are loaded in automatically if you have followed the instructions in Section C.1. Online help is available for these as for all R functions.

If you are having trouble with any of these functions, we suggest going to the website, `www.stat.columbia.edu/~gelman/arm/software/` and downloading the latest versions of everything.

The `lmer()` *function for multilevel modeling*

Our starting point for fitting multilevel models is `lmer()` ("linear mixed effects," but it also fits nonlinear models), a function that is currently part of the `lme4` package in R and can fit a variety of multilevel models using point estimation of variance parameters. We use `lmer()` for most of the examples in Part 2A of this book; as discussed in Section 16.1, `lmer()` is a good way to get quick approximate estimates before full multilevel modeling using Bugs. Various generalizations of `lmer()` are under development that would generalize it to perform fully Bayesian simulation-based inference; check the webpage `www.stat.columbia.edu/~gelman/arm/software/` for our links to the latest updates.

R packages

Go to the R webpage for information on R packages. To use a package, you must first install it (which can be done from the R console), then in any session you load in the package as needed using the `library()` function. Installation needs to be done only once, but you must load in the package with every R session. (You can load in our most frequently used packages automatically by putting lines into the `Rprofile.site` file, which is set up in the R directory on your computer if you follow the instructions in Section C.1.)

The most important packages for our purposes are `arm` (which has our own functions), `Matrix` and `lme4` (which include `lmer()` in its current form) and `R2WinBUGS` (which allows us to run Bugs from R, as described in the next section).

Other packages are helpful for specific purposes. For example, `hett` is a package that fits robust regression using the t model (see Section 6.6). To install, use the `install.packages()` function in R to download packages from the web. Then, in any session where we want to fit t regressions, for example, we type `library("hett")` (or include this line in any function call) and we are ready to go. To get help, we can click on Help at the top of the R window, then on "Html help," then on Packages, then on the package name (in this case, `hett`), then on the name of the function of interest (in this case, `tlm`). Alternatively, we can simply type `help(tlm)` or `?tlm` directly from the console.

Other R packages are available, and continue to be developed, to fit various complex models. The `MASS` package (which is automatically loaded if you follow the instructions in Section C.1) includes tools for fitting a variety of models. The `GAMM` package fits *generalized additive mixed models*, an adaptation of regression and generalized linear models that allows arbitrary nonlinear transformations of the input variables to be fit by the data (with "mixed" referring to the possibility of varying coefficients, that is, multilevel models); `sem` fits models for structural equations and instrumental variables (as shown in Section 10.6); and `MCMCpack` fits a variety of models, including multilevel linear regression for panel data.

C.3 Fitting models in Bugs and R

Calling Bugs from R

Currently, our main tool for fitting multilevel models is Bugs, which can be called from R using the `bugs()` function. (Type `?bugs` from the R console for more information.) As described in Part 2B of this book, we can use Bugs to fit models of essentially arbitrary complexity, but for large datasets or models with many parameters, Bugs becomes slow to converge.

Programming in R

The flexibility of Bugs makes it the preferred choice for now. If Bugs is too slow, or if it does not work for a particular model (yes, this happens!), then we program the Gibbs sampler and Metropolis algorithms directly in R. This will typically be faster than Bugs in computation time, and also it can converge in fewer iterations, because in programming the algorithm ourselves we have direct control and can use updating rules that are tailored to the particular model being fit. See Gelman et al. (2003, appendix C) for an example and Section 18.7 for an example using the Umacs package in R. (See `www.stat.columbia.edu/~gelman/arm/software/`.) If even R is too slow, the Gibbs and Metropolis algorithms can be programmed in Fortran

or C. Researchers are also developing compiled libraries for fast computation of multilevel models, linkable from R.

C.4 Fitting multilevel models using R, Stata, SAS, and other software

Several other programs are available to fit multilevel models. We shall briefly consider several popular packages, showing how they can be used to fit six prototype models.

We prefer R and Bugs for their flexibility, both in model fitting and in processing the resulting inferences, but we recognize that it is helpful to know how to fit multilevel models in software with which you are already familiar.

In addition to differences in syntax, the different packages display output differently. For example, we prefer to present estimated variance components in terms of standard deviations and (for varying-slope models) correlations, but some programs report variances and covariances. We shall assume that as a user of these other packages, you will be able to interpret the output and understand its relation to our notation in Section 2A of this book.

Six prototype models; fitting in R

We briefly present six example models along with the code needed to fit them in R using `lmer()`. The models can be fit in Bugs as described in Part 2B of this book. We follow with code in other packages. These examples do not come close to exhausting the kinds of multilevel models that we are fitting—but we hope they will be enough to get you started if you are using software other than R and Bugs.

1. Varying-intercept linear regression with data y, predictor x, and grouping variable `group`: $y_i = \alpha_{\text{group[i]}} + \beta x_i + \epsilon_i$ (that is, `group` is an index variable taking on integer values 1 through J, where J is the number of groups):

R code
```
lmer (y ~ x + (1 | group))
```

2. Same as example 1, but with a group-level predictor u (a vector of length J):

R code
```
u.full <- u[group]
lmer (y ~ x + u.full + (1 | group))
```

   (We need to define `u.full` to make a predictor that is the same length as the data; currently, the `lmer()` function in R does not take group-level predictors.)

3. Same as example 2, but with varying intercepts and varying slopes: $y_i = \alpha_{\text{group[i]}} + \beta_{\text{group[i]}} x_i + \epsilon_i$, where the J pairs (α_j, β_j) follow a bivariate normal distribution with mean vector $(\gamma_0^\alpha + \gamma_1^\alpha u_j,\ \gamma_0^\beta + \gamma_1^\beta u_j)$ and unknown 2×2 covariance matrix, with all parameters estimated from the data:

R code
```
lmer (y ~ x + u.full + x:u.full + (1 + x | group))
```

4. Go back to example 1, but with binary data and logistic regression: $\Pr(y_i = 1) = \text{logit}^{-1}(\alpha_{\text{group[i]}} + \beta x_i)$:

R code
```
lmer (y ~ x + (1 | group), family=binomial(link="logit"))
```

5. Go back to example 1, but with count data and overdispersed Poisson regression with offset $\log(z)$: $y_i \sim$ overdispersed Poisson$(z_i \exp(\alpha_{\text{group[i]}} + \beta x_i))$. This example includes overdispersion and an offset because both are important components to realistic count-data models. To fit quickly in R:

```
log.z <- log(z)                                                          R code
lmer (y ~ x + (1 | group),offset=log.z,family=quasipoisson(link="log"))
```

6. A two-way data structure with replication: for convenience, label the index variables for the groupings as `state` and `occupation`, so that the model is $y_i = \mu + \alpha_{\text{state[i]}} + \beta_{\text{occupation[i]}} + \gamma_{\text{state[i], occupation[i]}} + \epsilon_i$. We want the α's, the β's, and the γ's to be modeled (each with their own normal distribution); for simplicity, we assume no other predictors in the model. To fit:

```
state.occupation <- max(occupation)*(state - 1) + occupation              R code
lmer (y ~ 1 + (1 | state) + (1 | occupation) + (1 | state.occupation))
```

(The first line was needed to define an index variable that sweeps over all the states and occupations.)

Fitting in Stata

Stata (`www.stata.com`) is a statistical package that is particularly popular in social science and survey research. A wide range of multilevel models can be fit in Stata as extensions of the basic regression framework.

1. Varying-intercept linear regression:

```
xtmixed y x || group:                                                    Stata code
```

or

```
xtreg y x, i(group)                                                      Stata code
```

or

```
gllamm y x, i(group) adapt                                               Stata code
```

2. Varying-intercept linear regression with a group-level predictor:

   Stata has no concept of a vector of length shorter than the current dataset, so we have to create `ufull` and merge it with the dataset that includes `x` and `y`.

```
xtmixed y x ufull || group:                                              Stata code
```

or

```
xtreg y x ufull, i(group) re                                             Stata code
```

or

```
gen cons = 1                                                             Stata code
eq grp_c: cons
gllamm y x u, i(group) nrf(1) eqs(grp_c) adapt
```

3. Varying-intercept, varying-slope linear regression with a group-level predictor:

```
xtmixed y x ufull || group: x, cov(unstruct)                             Stata code
```

or

Stata code
```
gen cons = 1
eq grp_c: cons
eq grp_u: u
gllamm y x u, i(group) nrf(2) eqs(grp_c grp_u) adapt
```

4. Varying-intercept logistic regression:

Stata code
```
xtlogit y x, i(group)
```

or

Stata code
```
gllamm y x, i(group)family(binom)link(logit)
```

5. Varying-intercept overdispersed Poisson regression:

Stata code
```
xtnbreg y x, exposure(z) i(group)
```

Alternatively,

Stata code
```
xtnbreg y x, i(group) re offset(log.z)
```

or

Stata code
```
gllamm y x, i(group)offset(log.z) family(poi)link(log)
```

6. Varying-intercept linear regression with nested and non-nested groupings:

Stata code
```
egen state_occup = group(state occup)
xtmixed y || _all: R.state || _all: R.occup || _all: R.state_occup
```

Fitting in SAS

SAS (`www.sas.com`) is a statistical package that is particularly popular in biomedical research. As with Stata, many multilevel models can be fit in SAS by specifying grouping of the data.

1. Varying-intercept linear regression:

SAS code
```
proc mixed;
  class group;
  model y = x;
  random intercept / subject=group;
run;
```

2. Varying-intercept linear regression with a group-level predictor:

   SAS has no concept of a vector of length shorter than the current dataset, so we have to create `ufull` and merge it with the dataset that includes `x` and `y`.

SAS code
```
proc mixed;
  class group;
  model y = x ufull;
  random intercept / subject=group;
run;
```

3. Varying-intercept, varying-slope linear regression with a group-level predictor:

SAS code
```
proc mixed;
  class GROUP;
  model y = x ufull x*ufull;
  random intercept x / subject=group type=un;
run;
```

4. Varying-intercept logistic regression:

SAS code
```
proc nlmixed;
  parms b0 b1 s2;
  xbeta = b0 + b1*x + a;
  p = exp(xbeta)/(1+exp(xbeta));
  model y ~ binary(p);
  random a ~ normal(0,s2) subject = group;
run;
```

5. Varying-intercept overdispersed Poisson regression:

SAS code
```
proc nlmixed;
  parms b0=6 b1=-4 k=0.5 s2=1;
  xbeta = b0 + b1*x + a;
  mu = exp (logz + xbeta);
  p = mu/(mu+1/k);
  loglik = lgamma(y+1/k) - lgamma(y+1) - lgamma(1/k) +
    y*log(p) + (1/k)*log(1-p)
  model y ~ general(loglik);
  random a ~ normal(0,s2) subject = group;
run;
```

This `nlmixed` code specifies the negative binomial likelihood function; the parameters are then estimated by numerical integration. The `parms` statement names the parameters that will be estimated and gives starting values for them. The four lines that follow code the loglikelihood function that is going to be maximized. The `model` statement gives the response and the loglikelihood function, and the `random` statement defines the random intercept.

An alternative approach uses a preprogrammed negative binomial model:

SAS code
```
proc glimmix;
  class group;
  model y = x / solution dist = negbin offset = logz;
  random intercept / subject=group;
run;
```

The output of this run has a `scale` parameter (which equals k in the `nlmixed` code) to capture the overdispersion.

6. Varying-intercept linear regression with nested and non-nested groupings:

SAS code
```
proc mixed;
  class state occupation
  model y = ;
  random state occupation state*occupation;
run;
```

Fitting in SPSS

SAS (`www.spss.com`) is a statistical package that is particularly popular in psychology and experimental social science. Some multilevel models can be fit in SAS by specifying grouping in data.

1. Varying-intercept linear regression:

SPSS code

```
mixed
  y with x
 /fixed = x
 /print = solution testcov
 /random intercept | subject(group)
```

2. Varying-intercept linear regression with a group-level predictor:

   SPSS has no concept of a vector of length shorter than the current dataset, so we have to create `ufull` and merge it with the dataset that includes `x` and `y`.

SPSS code

```
mixed
  y with x ufull
 /fixed = x ufull
 /print = solution testcov
 /random intercept | subject(group)
```

3. Varying-intercept, varying-slope linear regression with a group-level predictor:

SPSS code

```
mixed
 y with x ufull
 /fixed = x ufull x*ufull
 /print = solution testcov
 /random intercept | subject(group) covtype(un)
```

We are not aware how to fit the other three examples (multilevel logistic regression, multilevel Poisson regression, and non-nested linear regression) in SPSS.

Fitting in AD Model Builder

AD Model Builder (`otter-rsch.com/admodel.htm`) is a package based on C++ that performs maximum likelihood or posterior simulation given the likelihood or posterior density function, a flexibility that is particularly helpful for nonlinear models.

1. Varying-intercept linear regression:

ADMB code

```
g = -0.5*norm2(z);
alpha = gamma_a + s(1)*z;
for (i=1;i<=n;i++)
  mu(i) = alpha(group(i)) + beta*x(i);
g += -n*log(s(0)) - 0.5*norm2((y-mu)/s(0));
```

   Here, `g` is the log-likelihood.

2. Varying-intercept linear regression with a group-level predictor:

ADMB code

```
g = -0.5*norm2(z);
alpha = gamma_a(0) + gamma_a(1)*u + s(1)*z;
for (i=1;i<=n;i++)
  mu(i) = alpha(group(i)) + beta*x(i);
g += -n*log(s(0)) - 0.5*norm2((y-mu)/s(0));
```

3. Varying-intercept, varying-slope linear regression with a group-level predictor:

ADMB code
```
g = -0.5*(norm2(z1)+norm2(z2));
alpha = gamma_a(0) + gamma_a(1)*u + s(1)*z1;
w = sqrt(1.0-square(rho));
beta = gamma_b(0) + gamma_b(1)*u + s(2)*(rho*z1 + w*z2);
for (i=1;i<=n;i++)
  mu(i) = alpha(group(i)) + beta(group(i))*x(i);
g += -n*log(s(0)) - 0.5*norm2((y-mu)/s(0));
```

4. Varying-intercept logistic regression:

ADMB code
```
g = -0.5*norm2(z);
alpha = gamma_a + s*z;
for (i=1;i<=n;i++)
  eta(i) = alpha(group(i)) + beta*x(i);
g += y*eta - sum(log(1+exp(eta)));
```

5. Varying-intercept overdispersed Poisson regression:

ADMB code
```
g = -0.5*norm2(z);
alpha = gamma_a + s*z;
for (i=1;i<=n;i++)
{
  lambda = offset(i)*exp(alpha(group(i)) + beta*x(i));
  omega = 1.0+lambda/kappa;
  g += log_negbinomial_density(y(i),lambda,omega);
}
```

   The negative binomial distribution may be viewed as an overdispersed Poisson distribution (with `omega` being the overdispersion coefficient).

6. Varying-intercept linear regression with nested and non-nested groupings:

ADMB code
```
g = -0.5*(norm2(z_a)+norm2(z_b)+norm2(z_g));
alpha = s(1)*z_a;
beta = s(2)*z_b;
gamma = s(3)*z_g;
for (i=1;i<=n;i++)
  eta(i) = mu + alpha(state(i)) + beta(occupation(i)) +
    gamma(state_occupation(i));
g += -n*log(s(0)) - 0.5*norm2((y-eta)/s(0));
```

Fitting in HLM, MLWin, and other software

HLM and MLWin are statistical programs specifically designed to fit multilevel models. They can fit models such as those in the preceding examples using a menu-based point-and-click approach.

One can also fit some or all of the models using other statistical packages, with varying degrees of difficulty. See here for an overview of many packages: `www.mlwin.com/softrev/index.html`; the descriptions there are not all up to date but they should provide a good starting point.

It is also possible to call Bugs using other software, including Stata, SAS, Python, Excel, and Matlab; go to the link at the Bugs homepage for "running from other software," currently at `www.mrc-bsu.cam.ac.uk/bugs/winbugs/remote14.shtml`

C.5 Bibliographic note

R (R Project, 2000) and Bugs (Spiegelhalter et al., 1994, 2002) have online help. In addition, Fox (2002) describes how to implement regressions in R, and Murrell (2005) shows R graphics. Becker, Chambers, and Wilks (1988) describes S, the predecessor to R. Venables and Ripley (2002) discuss statistical methods in R (or, essentially equivalently, S), focusing on nonparametric methods that are not covered here; the functions and examples used in that book are in the `MASS` package.

The `lmer()` function for fitting multilevel models is described by Bates (2005a, b), continuing on earlier work of Pinheiro and Bates (2000). Other R packages have been written for specific multilevel models; for example, `MCMCpack` (Martin and Quinn, 2002b).

For Bugs code, the books by Congdon (2001, 2003) present a series of examples. Kerman (2006) presents Umacs, and appendix C of Gelman et al. (2003) has examples of direct coding of Bayesian inference in R. The implementation of Bugs using R, as done in this book, is described by Sturtz, Ligges, and Gelman (2004). An open-source version of Bugs called OpenBugs (Thomas and O'Hara, 2005) is also under development.

Several software packages for multilevel models are reviewed by Centre for Multilevel Modelling (2005), including Stata, SAS, MLWin, and HLM. Rabe-Hesketh and Everitt (2003) is a good introduction to Stata, and Rabe-Hesketh and Skrondal (2005) describe how to fit multilevel models in Stata. The methods used by AD Model Builder are described by Fournier (2001) and Skaug and Fournier (2006).

Finally, various open-source software has been written and is under development for Bayesian inference and multilevel modeling; see, for example, Graves (2003), Plummer (2003), and Warnes (2003).