CHAPTER 17

Fitting multilevel linear and generalized linear models in Bugs and R

This chapter presents Bugs code for some of the multilevel models from Chapters 13–15, including varying intercepts and slopes, non-nested groupings, and multilevel versions of logistic regression and other generalized linear models.

17.1 Varying-intercept, varying-slope models

Simple model with no correlation between intercepts and slopes

We start with the varying-intercept, varying-slope radon model of Section 13.1, temporarily simplifying by ignoring the possible correlation between intercepts and slopes—that is, we model the intercepts and slopes as independent. Ignoring the multilevel correlation is inappropriate but can be helpful in getting started with the Bugs modeling.

Bugs code

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[county[i]] + b[county[i]]*x[i]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    b[j] ~ dnorm (b.hat[j], tau.b)
    a.hat[j] <- mu.a
    b.hat[j] <- mu.b
  }
  mu.a ~ dnorm (0, .0001)
  mu.b ~ dnorm (0, .0001)
  tau.a <- pow(sigma.a, -2)
  tau.b <- pow(sigma.b, -2)
  sigma.a ~ dunif (0, 100)
  sigma.b ~ dunif (0, 100)
}
```

This model looks a little more complicated than it needs to, in that we could have simply inserted `mu.a` and `mu.b` into the expressions for `a[j]` and `b[j]`. We include the intermediate quantities `a.hat[j]` and `b.hat[j]` because they become useful when the model includes correlations and group-level predictors, as we discuss in Section 17.2.

Modeling the correlation

We can write model (13.1) on page 279—which allows a correlation ρ between the varying intercepts and slopes—as follows:

Bugs code

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[county[i]] + b[county[i]]*x[i]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] <- B[j,1]
    b[j] <- B[j,2]
    B[j,1:2] ~ dmnorm (B.hat[j,], Tau.B[,])
    B.hat[j,1] <- mu.a
    B.hat[j,2] <- mu.b
  }
  mu.a ~ dnorm (0, .0001)
  mu.b ~ dnorm (0, .0001)

  Tau.B[1:2,1:2] <- inverse(Sigma.B[,])
  Sigma.B[1,1] <- pow(sigma.a, 2)
  sigma.a ~ dunif (0, 100)
  Sigma.B[2,2] <- pow(sigma.b, 2)
  sigma.b ~ dunif (0, 100)
  Sigma.B[1,2] <- rho*sigma.a*sigma.b
  Sigma.B[2,1] <- Sigma.B[1,2]
  rho ~ dunif (-1, 1)
}
```

Here we are using capital letters (`B`, `Tau`, `Sigma`) for matrix parameters and lowercase for vectors and scalars.

Scaled inverse-Wishart model

Another approach is to model the covariance matrix for the intercepts and slopes directly using the scaled inverse-Wishart distribution described at the end of Section 13.3. A useful trick with the scaling is to define the coefficients α_j, β_j in terms of multiplicative factors, ξ^α, ξ^β, with unscaled parameters $\alpha_j^{\text{raw}}, \beta_j^{\text{raw}}$ having the inverse-Wishart distribution. We first give the Bugs model, then explain it:

Bugs code

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[county[i]] + b[county[i]]*x[i]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] <- xi.a*B.raw[j,1]
    b[j] <- xi.b*B.raw[j,2]
    B.raw[j,1:2] ~ dmnorm (B.raw.hat[j,], Tau.B.raw[,])
    B.raw.hat[j,1] <- mu.a.raw
```

```
    B.raw.hat[j,2] <- mu.b.raw
  }
  mu.a <- xi.a*mu.a.raw
  mu.b <- xi.b*mu.b.raw
  mu.a.raw ~ dnorm (0, .0001)
  mu.b.raw ~ dnorm (0, .0001)

  xi.a ~ dunif (0, 100)
  xi.b ~ dunif (0, 100)

  Tau.B.raw[1:2,1:2] ~ dwish (W[,], df)
  df <- 3
  Sigma.B.raw[1:2,1:2] <- inverse(Tau.B.raw[,])
  sigma.a <- xi.a*sqrt(Sigma.B.raw[1,1])
  sigma.b <- xi.b*sqrt(Sigma.B.raw[2,2])
  rho <- Sigma.B.raw[1,2]/sqrt(Sigma.B.raw[1,1]*Sigma.B.raw[2,2])
}
```

The quantities that must be initialized—the parameters, in the terminology of Figure 16.4—include B^{raw}, the ξ's, the μ^{raw}'s, and T_B^{raw}. But we are actually interested in the derived quantities α, β, the μ's, the σ's, and ρ.

In the specification of the Wishart distribution for the inverse-variance `Tau.B.raw`, the matrix W is the prior scale, which we set to the identity matrix (in R, we write `W <- diag(2)`), and the degrees of freedom, which we set to 1 more than the dimension of the matrix (to induce a uniform prior distribution on ρ, as discussed near the end of Section 13.3). The call to the above model in R then looks something like

R code

```
W <- diag (2)
radon.data <- list ("n", "J", "y", "county", "x", "W")
radon.inits <- function (){
  list (B.raw=array(rnorm(2*J),c(J,2)), mu.a.raw=rnorm(1),
    mu.b.raw=rnorm(1), sigma.y=runif(1), Tau.B.raw=rwish(3,diag(2)),
    xi.a=runif(1), xi.b=runif(1))}
radon.parameters <- c ("a", "b", "mu.a", "mu.b", "sigma.y",
  "sigma.a", "sigma.b", "rho")
M1 <- bugs (radon.data, radon.inits, radon.parameters,
  "wishart1.bug", n.chains=3, n.iter=2000)
```

The advantage of this model, as compared to the simple model of σ_α, σ_β, and ρ presented on page 376, is that it generalizes more easily to models with more than two varying coefficients, as we discuss next. In addition, the extra parameters ξ_α, ξ_β can actually allow the computations to converge faster, an issue to which we shall return in Section 19.5 in the context of using redundant multiplicative parameters to speed computations for multilevel models.

Modeling multiple varying coefficients

As discussed in Section 13.3, when more than two coefficients are varying (for example, a varying intercept and two or more varying slopes), it is difficult to model the group-level correlations directly because of constraints involved in the requirement that the covariance matrix be positive definite. With more than two varying coefficients, it is simplest to just use the scaled inverse-Wishart model described above, using a full matrix notation to allow for an arbitrary number K of coefficients that vary by group.

Bugs code

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- inprod(B[county[i],],X[i,])
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    for (k in 1:K){
      B[j,k] <- xi[k]*B.raw[j,k]
    }
    B.raw[j,1:K] ~ dmnorm (mu.raw[], Tau.B.raw[,])
  }
  for (k in 1:K){
    mu[k] <- xi[k]*mu.raw[k]
    mu.raw[k] ~ dnorm (0, .0001)
    xi[k] ~ dunif (0, 100)
  }
  Tau.B.raw[1:K,1:K] ~ dwish (W[,], df)
  df <- K+1
  Sigma.B.raw[1:K,1:K] <- inverse(Tau.B.raw[,])
  for (k in 1:K){
    for (k.prime in 1:K){
      rho.B[k,k.prime] <- Sigma.B.raw[k,k.prime]/
        sqrt(Sigma.B.raw[k,k]*Sigma.B.raw[k.prime,k.prime])
    }
    sigma.B[k] <- abs(xi[k])*sqrt(Sigma.B.raw[k,k])
  }
}
```

And here is how the model could be called in R, assuming that the predictors (including the constant term) have already been bundled into a $n \times K$ matrix X:

R code

```
W <- diag (K)
data <- list ("n", "J", "K", "y", "county", "X", "W")
inits <- function (){
  list (B.raw=array(rnorm(J*K),c(J,K)), mu.raw=rnorm(K),
        sigma.y=runif(1), Tau.B.raw=rwish(K+1,diag(K)), xi=runif(K))}
parameters <- c ("B", "mu", "sigma.y", "sigma.B", "rho.B")
M2 <- bugs (radon.data, radon.inits, radon.parameters,
  "wishart2.bug", n.chains=3, n.iter=2000)
```

This reduces to our earlier model when $K = 2$.

Adding unmodeled individual-level coefficients

The above model allows all the coefficients B to vary. In practice it can be convenient to leave some regression coefficients unmodeled and let others vary by group. The simplest way to do this in the Bugs model is to add a vector β^0 of unmodeled coefficients for a matrix X^0 of predictors; the model is then $y_i = X_i^0\beta^0 + X_i B_{j[i]} + \epsilon_i$, with the fourth line of the Bugs model being written as

Bugs code

```
y.hat[i] <- inprod(b.0[],X.0[i,]) + inprod(B[county[i],],X[i,])
```

or, in one of the simpler models with only two varying coefficients,

Bugs code

```
y.hat[i] <- inprod(b.0[],X.0[i,]) + a[county[i]] + b[county[i]]*x[i]
```

In either case, if we define K^0 as the number of unmodeled coefficients (that is, the number of columns of X^0), we can specify a noninformative prior distribution at the end of the Bugs model:

Bugs code
```
for (k in 1:K.0){
  b.0[k] ~ dnorm (0, .0001)
}
```

17.2 Varying intercepts and slopes with group-level predictors

We can add group-level predictors to the varying-intercept, varying-slope Bugs models of the previous section by replacing `mu.a` and `mu.b` by group-level regressions.

Simplest varying-intercept, varying-slope model. For example, we can add a group-level predictor `u` to the very first model of this chapter by replacing the expressions for `a.hat[j]` and `b.hat[j]` with

Bugs code
```
a.hat[j] <- g.a.0 + g.a.1*u[j]
b.hat[j] <- g.b.0 + g.b.1*u[j]
```

and then removing the prior distributions for `mu.a` and `mu.b` and replacing with `dnorm (0, .0001)` prior distributions for each of `g.a.0, g.a.1, g.b.0`, and `g.b.1`.

Model in which intercepts and slopes are combined into a matrix, B. A similar operation works with the model on page 376 also, except that `a.hat[j]`, `b.hat[j]` become `B.hat[j,1]`, `B.hat[j,2]`.

Scaled inverse-Wishart model. In the scaled model on page 376, we add a group-level predictor by replacing `mu.a.raw` with `g.a.0.raw + g.a.1.raw*u[j]` and similarly for `mu.b.raw`. We continue by multiplying the raw parameters for `a` and `b` by `xi.a` and `xi.b`, respectively to get `g.a.0, g.a.1, g.b.0, g.b.1`.

Multiple group-level predictors

If we have several group-level predictors (expressed as a matrix U with J rows, one for each group), we use notation such as

Bugs code
```
a.hat[j] <- inprod (g.a[], U[j,])
b.hat[j] <- inprod (g.b[], U[j,])
```

with a loop setting up the prior distribution for the elements of `g.a` and `g.b`.

Multiple varying coefficients with multiple group-level predictors

In a more general notation, we can express both the individual-level and group-level regression using matrices, generalizing the model on page 377 to:

Bugs code
```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- inprod(B[county[i],],X[i,])
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (k in 1:K){
    for (j in 1:J){
```

```
      B[j,k] <- xi[k]*B.raw[j,k]
    }
    xi[k] ~ dunif (0, 100)
  }
  for (j in 1:J){
    B.raw[j,1:K] ~ dmnorm (B.raw.hat[j,], Tau.B.raw[,])
    for (k in 1:K){
      B.raw.hat[j,k] <- inprod(G.raw[k,],U[j,])
    }
  }
  for (k in 1:K){
    for (l in 1:L){
      G[k,l] <- xi[k]*G.raw[k,l]
      G.raw[k,l] ~ dnorm (0, .0001)
    }
  }
  Tau.B.raw[1:K,1:K] ~ dwish (W[,], df)
  df <- K+1
  Sigma.B.raw[1:K,1:K] <- inverse(Tau.B.raw[,])
  for (k in 1:K){
    for (k.prime in 1:K){
      rho.B[k,k.prime] <- Sigma.B.raw[k,k.prime]/
        sqrt(Sigma.B.raw[k,k]*Sigma.B.raw[k.prime,k.prime])
    }
    sigma.B[k] <- abs(xi[k])*sqrt(Sigma.B.raw[k,k])
  }
}
```

Adding unmodeled individual-level coefficients

One can further extend these models by adding a matrix of predictors X^0 with unmodeled coefficients β^0 as described at the end of Section 17.1. In Bugs, this is done by adding `inprod(b.0[],X.0[i,])` to the expression for `y.hat[i]`.

17.3 Non-nested models

Here is the Bugs code for model (13.9) on page 289, in which the data are grouped in two different ways (by treatment and airport in the flight simulator example):

Bugs code

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- mu + gamma[treatment[i]] + delta[airport[i]]
  }
  mu ~ dnorm (0, .0001)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:n.treatment){
    gamma[j] ~ dnorm (0, tau.gamma)
  }
  tau.gamma <- pow(sigma.gamma, -2)
  sigma.gamma ~ dunif (0, 100)

  for (k in 1:n.airport){
```

```
    delta[k] ~ dnorm (0, tau.delta)
  }
  tau.delta <- pow(sigma.delta, -2)
  sigma.delta ~ dunif (0, 100)
}
```

With non-nested models, there is a choice of where to put the intercept or constant term: here we have included it as μ in the data model, with zero means for the group effects, but another option would be to include a mean parameter for the γ_j's or δ_k's. Including a constant term in more than one place would lead to nonidentifiability. As we discuss in Section 19.4, it can be useful for both conceptual and computational reasons to include redundant mean parameters and then reparameterize the model to regain identifiability.

17.4 Multilevel logistic regression

Multilevel generalized linear models have the same structure as linear models, altering only the data model. To illustrate, we show the logistic regression for state-level opinions from Section 14.1. The model as written in Bugs looks more complicated than it really is, because we must explicitly specify distributions for all the parameters and we must transform from standard-deviation parameters σ to inverse-variances τ. Here is the model:

Bugs code

```
model {
  for (i in 1:n){
    y[i] ~ dbin (p.bound[i], 1)
    p.bound[i] <- max(0, min(1, p[i]))
    logit(p[i]) <- Xbeta[i]
    Xbeta[i] <- b.0 + b.female*female[i] + b.black*black[i] +
      b.female.black*female[i]*black[i] + b.age[age[i]] + b.edu[edu[i]] +
      b.age.edu[age[i],edu[i]] + b.state[state[i]]
  }
  b.0 ~ dnorm (0, .0001)
  b.female ~ dnorm (0, .0001)
  b.black ~ dnorm (0, .0001)
  b.female.black ~ dnorm (0, .0001)

  for (j in 1:n.age) {b.age[j] ~ dnorm (0, tau.age)}
  for (j in 1:n.edu) {b.edu[j] ~ dnorm (0, tau.edu)}
  for (j in 1:n.age) {for (k in 1:n.edu){
    b.age.edu[j,k] ~ dnorm (0, tau.age.edu)}}
  for (j in 1:n.state) {
    b.state[j] ~ dnorm (b.state.hat[j], tau.state)
    b.state.hat[j] <- b.region[region[j]] + b.v.prev*v.prev[j]}
  b.v.prev ~ dnorm (0, .0001)
  for (j in 1:n.region) {b.region[j] ~ dnorm (0, tau.region)}

  tau.age <- pow(sigma.age, -2)
  tau.edu <- pow(sigma.edu, -2)
  tau.age.edu <- pow(sigma.age.edu, -2)
  tau.state <- pow(sigma.state, -2)
  tau.region <- pow(sigma.region, -2)

  sigma.age ~ dunif (0, 100)
  sigma.edu ~ dunif (0, 100)
```

```
  sigma.age.edu ~ dunif (0, 100)
  sigma.state ~ dunif (0, 100)
  sigma.region ~ dunif (0, 100)
}
```

The "logistic regression" part of this model is at the beginning. The data distribution, `dbin`, is the *binomial distribution* with $N = 1$, meaning that $y_i = 1$ with probability p_i and 0 otherwise. The quantity `p.bound` is defined to restrict the probability to lie between 0 and 1, a trick we use to keep Bugs from crashing. The model continues with a usual multilevel formulation, with the only new feature being the nested loops for the matrix parameter `b.age.edu`. The model is actually slow to converge under this parameterization, but in improving it we shall keep this basic structure (see Section 19.4 for details).

17.5 Multilevel Poisson regression

Poisson models are straightforward to code in Bugs. Here we show model (15.1) from page 326 for the analysis of the police stops. This model includes crossed multilevel predictors and overdispersion:

Bugs code

```
model {
  for (i in 1:n){
    stops[i] ~ dpois (lambda[i])
    log(lambda[i]) <- offset[i] + mu +
      b.eth[eth[i]] + b.precinct[precinct[i]] + epsilon[i]
    epsilon[i] ~ dnorm (0, tau.epsilon)
  }
  mu ~ dnorm (0, .0001)
  mu.adj <- mu + mean(b.eth[]) + mean(b.precinct[])
  tau.epsilon <- pow(sigma.epsilon, -2)
  sigma.epsilon ~ dunif (0, 100)

  for (j in 1:n.eth){
    b.eth[j] ~ dnorm (0, tau.eth)
    b.eth.adj[j] <- b.eth[j] - mean(b.eth[])
  }
  tau.eth <- pow(sigma.eth, -2)
  sigma.eth ~ dunif (0, 100)

  for (j in 1:n.precinct){
    b.precinct[j] ~ dnorm (0, tau.precinct)
    b.precinct.adj[j] <- b.precinct[j] - mean(b.precinct[])
  }
  tau.precinct <- pow(sigma.precinct, -2)
  sigma.precinct ~ dunif (0, 100)
}
```

As in (15.1), the ϵ_i's represent overdispersion, and σ_ϵ is a measure of the amount of overdispersion in the data. We computed the offset term in R (as $\log(\frac{15}{12}n)$) and included it in the data in the `bugs()` call. (Alternatively, we could have computed the offset directly within the initial loop of the Bugs model.) Finally, the adjusted parameters `mu.adj` and `b.eth.adj` correspond to μ' and α'_e in equations (15.2) and (15.3). For completeness we have adjusted the precinct intercepts to sum to zero also. (This adjustment is less important because there are 75 of them, so their mean will be close to zero in any case.)

17.6 Multilevel ordered categorical regression

Sections 6.5 and 15.2 describe an ordered categorical logistic regression. Here we show the model as written in Bugs. The data-level model is adapted from an example from the online Bugs manual, and the rest of the Bugs code describes the particular model we fit to the storable-votes data. In this model, `dcat` is the Bugs notation for the distribution defined by probabilities `p[i,1], p[i,2], p[i,3]` of observation y_i falling in each of the 3 categories, and these probabilities p are defined in terms of the cumulative probabilities Q, which follow a logistic regression.

Bugs code

```
model {
  for (i in 1:n){
    y[i] ~ dcat(P[i,])
    P[i,1] <- 1 - Q[i,1]
    for (i.cut in 2:n.cut){
      P[i,i.cut] <- Q[i,i.cut-1] - Q[i,i.cut]
    }
    P[i,n.cut+1] <- Q[i,n.cut]
    for (i.cut in 1:n.cut){
      logit(Q[i,i.cut]) <- z[i,i.cut]
      Z[i,i.cut] <- (x[i] - C[player[i],i.cut])/s[player[i]]
    }
  }
  for (i.player in 1:n.player){
    C[i.player,1] ~ dnorm (mu.c[1], tau.c[1])I(0,C[i.player,2])
    C[i.player,2] ~ dnorm (mu.c[2], tau.c[2])I(C[i.player,1],100)
    s[i.player] ~ dlnorm (mu.log.s, tau.log.s)I(1,100)
  }
  for (i.cut in 1:n.cut){
    mu.c[i.cut] ~ dnorm (0, 1.E-6)
    tau.c[i.cut] <- pow(sigma.c[i.cut], -2)
    sigma.c[i.cut] ~ dunif (0, 1000)
  }
  mu.log.s ~ dnorm (0, .0001)
  tau.log.s <- pow(sigma.log.s, -2)
  sigma.log.s ~ dunif (0, 1000)
}
```

We continue with the notation in which matrices (in this case, `P`, `Q`, `Z`, and `C`) are written as capital letters, with lowercase used for vectors and scalars.

The above model uses bounds (the `I(,)` notation) for two purposes. First, the bounds `c[i.player,1]` and `c[i.player,2]` constrain these two parameters for each player to fall between 0 and 100 (see the discussion following model (6.11) on page 121), and to be in increasing order, which is necessary so that the probability for each category is nonnegative.

The second use of bounds in the Bugs model is to constrain each `s[i.player]` to fall between 1 and 100, which we do for purely computational reasons. When we allowed these parameters to float freely, Bugs would crash. We suspect that Bugs was crashing because the ratios calculated for the logistic distribution were too extreme, and constraining the parameters `s` stopped this problem. Another way to constrain this in Bugs would be to parameterize in terms of $1/s$.

17.7 Latent-data parameterizations of generalized linear models

Logistic regression

As discussed in Section 5.3, logistic regression can be expressed directly or using latent parameters. We can similarly implement both formulations in Bugs.

Section 17.4 shows an example of the direct parameterization. The equivalent latent-data version begins as follows:

Bugs code

```
model {
  for (i in 1:n){
    z.lo[i] <- -100*equals(y[i],0)
    z.hi[i] <- 100*equals(y[i],1)
    z[i] ~ dlogis (Xbeta[i], 1) I(z.lo[i], z.hi[i])
```

After this, the two models are the same. For each data point, we have simply defined the latent variable z_i and restricted it to be positive if $y_i = 1$ and negative if $y_i = 0$. (For a logistic model, restricting z_i to the range $(0, 100)$ is essentially equivalent to restricting to $(0, \infty)$.)

In calling the model from R, it is helpful to initialize the z_i's along with the other parameters in the model. We want to use random numbers, but they must respect the restriction that z be positive if and only if $y = 1$. We can do this by adding the following, within the list *inside* the `inits()` function:

R code

```
z=runif(n)*ifelse(y==1,1,-1)
```

This gives initial values for z in the range $(0, 1)$ if $y = 1$, or the range $(-1, 0)$ if $y = 0$.

Robit regression

Section 6.6 describes a robust alternative to logistic regression constructed by replacing the logistic latent-data distribution with the t. We can implement robit regression in Bugs in three steps. First, within the data model (in the `for (i in 1:n)` loop, we use the t distribution:

Bugs code

```
    z[i] ~ dt (z.hat[i], tau.z, df) I(z.lo[i], z.hi[i])
```

Second, outside the data loop, we scale the t distribution as indicated in (6.15) on page 125, so that its variance equals 1 for any value of the degrees-of-freedom parameter:

Bugs code

```
tau.z <- df/(df-2)
```

Third, we give the inverse-variance parameter a uniform prior distribution from 0 to 0.5, which has the effect of restricting the degrees of freedom to be at least 2 (a constraint required by Bugs):

Bugs code

```
df <- 1/df.inv
df.inv ~ dunif (0, .5)
```

In addition, we must initialize z within the `inits()` function as described previously for latent-data logistic regression. The probit model can be implemented in a similar way.

17.8 Bibliographic note

The Gibbs sampler and Metropolis algorithm were first presented by Metropolis et al. (1953), for applications in statistical physics. Tanner and Wong (1987), Gelfand and Smith (1990), and Gelfand et al. (1990) demonstrated the application of these computational ideas to general statistical models, including hierarchical linear regression. Zeger and Karim (1991), Karim and Zeger (1992), Albert and Chib (1993), Dellaportas and Smith (1993), and others developed Gibbs sampler and Metropolis algorithms for hierarchical generalized linear models. Chen, Shao, and Ibrahim (2000) and Liu (2002) review more advanced work on iterative simulation.

The augmented-data formulation of multilevel regression appears in Lindley and Smith (1972) and Hodges (1998). Gelman et al. (2003) derive the algebra of Bayesian multilevel modeling and discuss partial pooling and the Gibbs sampler in detail; chapters 11 and 15 of that book discuss computation for multilevel models. The general references on multilevel modeling (see the note at the end of Chapter 12) are relevant here. See appendix C of Gelman et al. (2003) for more on programming the Gibbs sampler and related simulation algorithms in R.

17.9 Exercises

1. Parameterizing varying-intercept, varying-slope models: the folder `nes` contains data from the National Election Study surveys. Set up a model for party identification (as a continuous outcome, as in Section 4.7), given the predictors shown in Figure 4.6 on page 74, and also allowing the intercept and the coefficient for ideology to vary by state. You will fit various versions of this model (using data from the year 2000) in Bugs.
   (a) Fit the model with no correlation between the intercepts and the slopes (the coefficients for ideology).
   (b) Fit the model with a uniform prior distribution on the correlation between the intercepts and slopes.
   (c) Fit the scaled inverse-Wishart model.
   (d) Compare the inferences from the three models.
2. Understanding and summarizing varying-intercept, varying-slope models: continue the example from the previous exercise.
   (a) Add, as a group-level predictor, the average income by state. Discuss how the parameter estimates change when this predictor has been added.
   (b) Graph the fitted model as in Figures 13.1–13.2 on page 281.
3. Multiple varying coefficients:
   (a) Repeat the previous exercise, this time allowing the coefficient for income to vary by state as well.
   (b) Repeat, allowing all the coefficients to vary by state.
   (c) Summarize your inferences from these models graphically.
4. Fitting non-nested models: use Bugs to fit the model for Olympic judging from Exercises 13.3.
5. Models with unequal variances: use Bugs to fit the model you set up for the age-guessing data in Exercise 13.4.
6. Multilevel logistic regression: use Bugs to fit the model for the well-switching data from Exercise 14.2.

7. Fitting a choice model: set up a model for the arsenic decision problem, as described near the end of Section 6.8, modeling a distribution for the parameters (a_i, b_i, c_i) in the population.
8. Elements of a Bugs model:
   (a) List the elements of the model on page 379 by category: modeled data, unmodeled data, modeled parameters, unmodeled parameters, derived quantities, and looping indexes (as in Figure 16.4).
   (b) Do the same for the model on page 381.
   (c) Do the same for the model on page 383.
9. Ordered logit: consider the ordered logistic regression for vote intention in Exercise 15.1.
   (a) Fit a classical ordered logistic regression (without the coefficients for states) using `lmer()`.
   (b) Fit the classical ordered logistic regression using Bugs. Compare to the estimate from (a).
10. Multilevel ordered logit:
   (a) Take the previous exercise and allow the intercepts to vary by state.
   (b) Plot the fitted model from (a), along with the model from Exercise 17.9(a), for a set of eight states (as in Figure 14.2 on page 307, but for this three-category model) and discuss how they differ.
   (c) Discuss whether it is worth fitting this model, as compared to a logistic model that includes only two categories, discarding the respondents who express no opinion or support other candidates.
11. Multilevel ordered logit, probit, and robit models: the folder `storable` has data from the storable-votes experiment described in Sections 6.5, 15.2, and 17.6.
   (a) Fit an ordered logistic regression for the three categories, with a different intercept for each person.
   (b) Fit the same model but using the probit link.
   (c) Fit the same model but using the robit link.
   (d) Compare the estimates from the three models. The coefficients might not be comparable, so you have to plot the fitted models as in Figure 6.4 on page 121.
12. Multivariate outcomes: the folder `beta.blockers` contains data from a meta-analysis of 22 clinical trials of beta-blockers for reducing mortality after myocardial infarction (from Yusuf et al., 1985; see also Carlin, 1992, and Gelman et al., 2003, chapter 5).
   (a) Set up and fit a logistic regression model to estimate the effect of the treatment on the probability of death, allowing different death rates and different treatment effects for the different studies. Summarize by the estimated treatment average effect and its standard error.
   (b) Make a graph or graphs displaying the data and fitted model.