Stat 532 Assignment 8 (Part 2)

Kenny Flagg

October 30, 2015

2. (a) In his paper, Gelman proposed a new way of doing ANOVA based on defining a batch of effects corresponding to each level of variation and estimating the variance of the effects in each batch. I agree that this is a more intuitive and understandable way to perform ANOVA, but I am not convinced that it is as novel or as simple as he claims. Is there truly a difference between defining batches and defining a nesting structure? Is the "automatic" creation of the variance structure really automatic in a practical sense?

   Gelman tried to separate the model from the design by thinking about batches. He commented that "the variance components and standard errors are estimated from the data, without any need to specify comparisons based on the design" but all variables in the design must be included for the model to have the correct structure. I see no difference between thinking about batches and thinking about the nesting of the design. The correct structure my not be obvious, but it can be found by thinking carefully about each variable in the design and deciding what level it acts on. When there is no nesting, as in the connection times example, the new ANOVA table based on batches has the same structure as the traditional ANOVA table. Gelman has defined a new way of thinking, not a new way of modeling.

   I take issue with his use of the word "automatic" throughout the paper. The variance structure is defined automatically by the batches, but performing the analysis still requires defining the structure in a way that the software can work with. For both a non-statistician using canned software, and for a statistician using a Bayesian hierarchical model, this will typically mean translating the batch structure into a old-fashioned hierarchy. Defining the batches and defining the hierarchical variance structure are both tasks that require non-trivial thought, so I would argue that this is not automatic in any sense and I would love to ask Gelman what he thinks the word means to his audience.

   Despite the issue I discussed above, I do like this new ANOVA paradigm. The graphical presentation of standard errors in the ANOVA table is vastly more informative than the traditional table of mean squares. Estimating both the finite-population and infinite-population variances provides very useful flexibility. The terms "varying" and "constant" are more meaningful than "random" and "fixed". I will become an advocate for Gelman's ANOVA table.

   (b) I find it most natural to think directly about infection rates, and to think of effects as multiplicative. I defined infection rates for each variety and then scaled them to represent the effects of other variables. For example, if variety 1 has an infection rate of p in block A, and block B has 1.2 times as many virus-carrying mites as block A, then the

infection rate for variety 1 in block B should be $1.2p$. I use Beta distributions to model probabilities, and Uniform distributions to model effects. I chose Uniform distributions to constrain the effects to reasonable sizes but also to represent uncertainty about what distribution they would really follow.
The first bit of code defined the variables used later. I report this to give context to the rest of the code.

```
set.seed(8723)

# Number of leaves collected from each plot
nLeaves <- 30

# Inoculation status
nStatus <- 2
statusNames <- c('INOC', 'CNTL')

# Nitrogen application timing treatments
nTrts <- 3
trtNames <- c('Early', 'Mid', 'Late')

# Varieties
nVars <- 5
varNames <- paste('Variety', 1:nVars, sep = '')

# Blocks
nBlocks <- 6
blockNames <- LETTERS[1:nBlocks]

# Total number of observtions
n <- nBlocks * nVars * nTrts * nStatus
```

First consider inoculated plots, where there are interesting interactions. There is a small amount of block-to-block variation due to the number of virus-carrying mites in the area. This causes rates to be multiplied by a Unif(0.8, 1.2) block effect that does not interact with variety or status (but the researchers would like to account for those interactions).

```
block.eff <- runif(nBlocks, 0.8, 1.2)
names(block.eff) <- blockNames
```

Varieties 1 and 2 are from a very resistant population with mean rate of $\frac{1}{20}$ and a lot of mass near 0. Varieties 3, 4, and 5 are from a susceptible population with mean rate of $\frac{3}{20}$ but the rates are definitely positive.

```
var.inoc.eff <- c(rbeta(2, 1, 19), rbeta(3, 3, 17))
names(var.inoc.eff) <- varNames
```

Infection rate is halved for early treatment, 75% for mid treatment, and unaffected by late treatment. Treatment and variety do not interact.

```
trt.inoc <- c(0.5, 0.75, 1)
names(trt.inoc) <- trtNames
```

Now consider uninoculated plots, where there is not much infection because there are few mites present. I'll use the same block adjustments as before (no block by status interaction). Base infection rates for the varieties are lower in control plots. The resistant

population still has mass near 0 and the susceptible population still has nonzero infection rate.

```
var.cntl.eff <- c(rbeta(2, 1, 39), rbeta(3, 3, 37))
names(var.cntl.eff) <- varNames
```

Treatment is less effective because there isn't as much virus exposure.

```
trt.cntl <- c(0.8, 0.9, 1)
names(trt.cntl) <- trtNames
```

Next, I computed the "true" infection rate for each plot by multiplying the effects.

```
# It makes the most sense to me to store the infection rates in an array.
probs <- array(numeric(0), dim = c(nTrts, nVars, nBlocks, nStatus))
dimnames(probs) <- list(trtNames, varNames, blockNames, statusNames)

for(b in blockNames){
  for(v in varNames){
    # Store inoculated rates
    probs[, v, b, 2] <- block.eff[b] * var.inoc.eff[v] * trt.inoc

    # Store control rates
    probs[, v, b, 1] <- block.eff[b] * var.cntl.eff[v] * trt.cntl
  }
}
```

Finally, I created a data frame and drew appropriate binomial values.

```
# Initialize a data.frame
data.sim <- data.frame(
  'infected' = as.numeric(NA),
  expand.grid(
    'status' = factor(statusNames),
    'nitrogen' = factor(trtNames),
    'variety' = factor(varNames),
    'block' = factor(blockNames)
    )
  )

# Loop through the data frame and generate draws!
for(i in 1:n){
  data.sim$infected[i] <- with(
    data.sim,
    rbinom(1, nLeaves,
           probs[
             nitrogen[i],
             variety[i],
             block[i],
             status[i]
             ]
           )
    )
}
```

Figure 1 presents the simulated data in a heatmap.

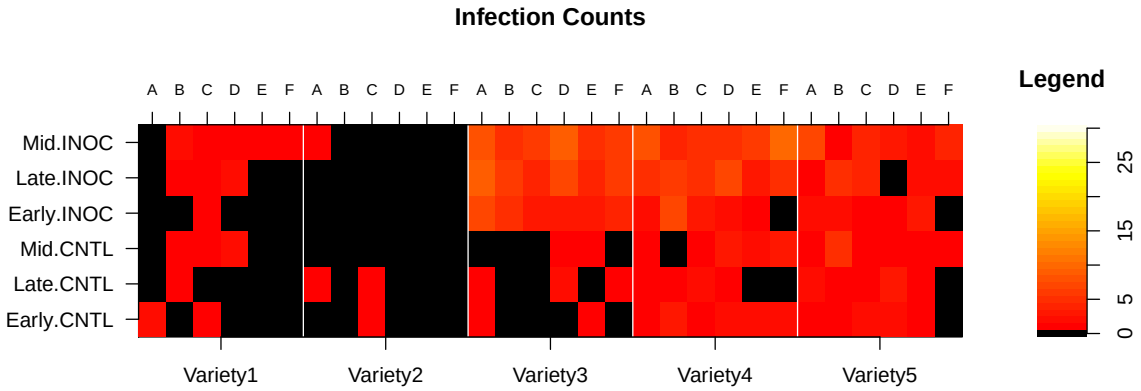

Figure 1: Simulated infection counts.

(c) The data generating process has a clear hierarchy that does not entirely resemble the experiment design. Infection occurs in control plots through a different process than in inoculated plots. The first batch of coefficients describes the status of being inoculated or control. The next most general level is that of block, so the second batch corresponds to the block effects within inoculated/control status. The third batch is for variety within block, and the final batch is for treatment within variety.

The model is

$$\begin{aligned} y_{ijkl} &\sim \text{Binomial}(30, \pi_{ijkl}), \\ \pi_{ijkl} &= b_j \phi_{ik} t_{il}, \\ b_j &\sim \text{Unif}(0.8, 1.2) \text{ is the effect of block } j. \end{aligned}$$

For control plots,

$$\begin{aligned} \phi_{1k} | \text{variety } k \text{ is resistant} &\sim \text{Beta}(1, 39), \\ \phi_{1k} | \text{variety } k \text{ is susceptible} &\sim \text{Beta}(3, 37), \\ t_{11} = 0.8, t_{12} = 0.9, t_{13} = 1 &\text{ are the treatment effects.} \end{aligned}$$

For inoculated plots,

$$\begin{aligned} \phi_{2k} | \text{variety } k \text{ is resistant} &\sim \text{Beta}(1, 19), \\ \phi_{2k} | \text{variety } k \text{ is susceptible} &\sim \text{Beta}(3, 17), \\ t_{21} = 0.5, t_{22} = 0.75, t_{23} = 1 &\text{ are the treatment effects.} \end{aligned}$$

I simulated the data assuming it was known which population each variety came from. If this is unknown, the analysis should include a prior distribution for the probability that a variety is resistant or susceptible. Most researchers would default to using a Binomial GLMM for this experiment, which is reasonable, but they would probably think of inoculation as a treatment, which leads to an incorrect nesting structure.

3. (a) The model is

$$\begin{aligned} y_i|\mu, V_i &\sim \mathrm{N}(\mu, V_i), \\ \mu &\sim \mathrm{Unif}(-\infty, \infty), \\ V_i|\sigma^2 &\sim \mathrm{Inv}{-}\chi^2(\nu, \sigma^2), \\ \log(\sigma) &\sim \mathrm{Unif}(0, \infty) \end{aligned}$$

where $i = 1, \ldots, n$, and n and ν are known. The goal is to find the posterior distribution $p(\mu, \sigma^2|y, V)$.

For data, I drew the following 10 values from a $t_{\nu=9}(\mu = 20, \sigma^2 = 3^2)$ distribution.

12.194 19.043 19.84 23.882 15.634 24.578 24.975 19.785 21.103 17.964

(b) Most of this code sets up the variables. The actual sampler is short! I sampled for 300 iterations and threw out 50 as warmup. I plotted the posterior distributions in Figure 2, which appears after the code.

```
n.chains <- 4
n.iter <- 300
set.seed(9723)

# Initialize a list of lists to hold the draws
gibbs1 <- replicate(n.chains, simplify = FALSE,
                            list('V' = matrix(nrow = n.iter, ncol = n),
                                 'mu' = numeric(n.iter),
                                 'sigma' = numeric(n.iter)))

# Initial values for chain 1
gibbs1[[1]]$V[1,] <- 0.25
gibbs1[[1]]$mu[1] <- 8
gibbs1[[1]]$sigma[1] <- 0.5

# Initial values for chain 2
gibbs1[[2]]$V[1,] <- 0.25
gibbs1[[2]]$mu[1] <- 32
gibbs1[[2]]$sigma[1] <- 0.5

# Initial values for chain 3
gibbs1[[3]]$V[1,] <- 100
gibbs1[[3]]$mu[1] <- 8
gibbs1[[3]]$sigma[1] <- 10

# Initial values for chain 4
gibbs1[[4]]$V[1,] <- 100
gibbs1[[4]]$mu[1] <- 32
gibbs1[[4]]$sigma[1] <- 10

# Complete conditionals
draw.V <- function(y, n, nu, mu, sigma){
  return(1/rgamma(n, (nu+1)/2, (nu*sigma^2+(y-mu)^2)/2))
}
draw.mu <- function(y, V){
  return(rnorm(1, sum(y/V)/sum(1/V), 1/sqrt(sum(1/V))))
}
```

```
draw.sigma <- function(n, nu, V){
  return(sqrt(rgamma(1, n*nu/2, nu*sum(1/V)/2)))
}

# Outer loop for the iterations
for(i in 2:n.iter){
  # Inner loop for the chains
  for(j in 1:n.chains){
    # V
    gibbs1[[j]]$V[i,] <- with(gibbs1[[j]], draw.V(y, n, nu, mu[i-1], sigma[i-1]))
    # Mu
    gibbs1[[j]]$mu[i] <- draw.mu(y, gibbs1[[j]]$V[i,])
    # Sigma
    gibbs1[[j]]$sigma[i] <- draw.sigma(n, nu, gibbs1[[j]]$V[i,])
  }
}
```

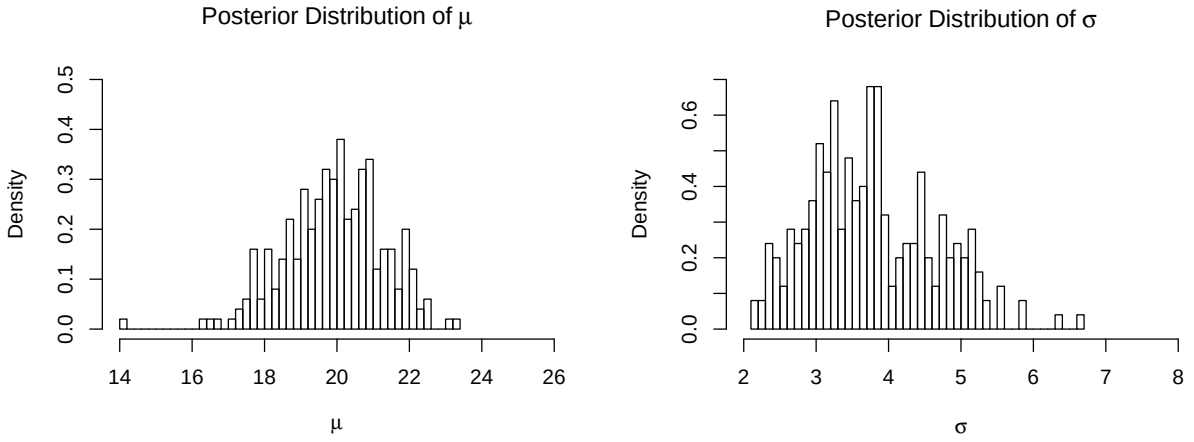

Figure 2: Posterior distributions from the augmented model.

(c) This version is no more difficult to implement than the version in part (b). I ran it for 300 iterations, discarded the first 20, and plotted the posterior distributions in Figure 3.

```
n.chains <- 4
n.iter <- 300
set.seed(35255)

# Initialize a list of lists to hold the draws
gibbs2 <- replicate(n.chains, simplify = FALSE,
                            list('U' = matrix(nrow = n.iter, ncol = n),
                                 'alpha' = numeric(n.iter),
                                 'mu' = numeric(n.iter),
                                 'tau' = numeric(n.iter),
                                 'sigma' = numeric(n.iter)))

# Initial values for chain 1
gibbs2[[1]]$U[1,] <- 0.25
```

```
gibbs2[[1]]$alpha[1] <- 1
gibbs2[[1]]$mu[1] <- 8
gibbs2[[1]]$tau[1] <- 0.5
gibbs2[[1]]$sigma[1] <- 0.5

# Initial values for chain 2
gibbs2[[2]]$U[1,] <- 0.25
gibbs2[[2]]$alpha[1] <- 1
gibbs2[[2]]$mu[1] <- 32
gibbs2[[2]]$tau[1] <- 0.5
gibbs2[[1]]$sigma[1] <- 0.5

# Initial values for chain 3
gibbs2[[3]]$U[1,] <- 100
gibbs2[[3]]$alpha[1] <- 1
gibbs2[[3]]$mu[1] <- 8
gibbs2[[3]]$tau[1] <- 10
gibbs2[[1]]$sigma[1] <- 10

# Initial values for chain 4
gibbs2[[4]]$U[1,] <- 100
gibbs2[[4]]$alpha[1] <- 1
gibbs2[[4]]$mu[1] <- 32
gibbs2[[4]]$tau[1] <- 10
gibbs2[[1]]$sigma[1] <- 10

# Complete conditionals
draw.U <- function(y, n, nu, mu, tau, alpha){
  return(1/rgamma(n, (nu+1)/2, (nu*tau^2+((y-mu)/alpha)^2)/2))
}
draw.mu <- function(y, U, alpha){
  return(rnorm(1, sum(y/(U*alpha^2))/sum(1/(U*alpha^2)), 1/sqrt(sum(1/(U*alpha^2)))))
}
draw.tau <- function(n, nu, U){
  return(sqrt(rgamma(1, n*nu/2, nu*sum(1/U)/2)))
}
draw.alpha <- function(n, mu, U){
  return(1/sqrt(rgamma(1, n/2, sum((y-mu)^2/U)/2)))
}

# Outer loop for the iterations
for(i in 2:n.iter){
  # Inner loop for the chains
  for(j in 1:n.chains){
    # U
    gibbs2[[j]]$U[i,] <- with(gibbs2[[j]], draw.U(y, n, nu, mu[i-1], tau[i-1], alpha[i-1]))
    # Mu
    gibbs2[[j]]$mu[i] <- draw.mu(y, gibbs2[[j]]$U[i,], gibbs2[[j]]$alpha[i-1])
    # Tau
    gibbs2[[j]]$tau[i] <- draw.tau(n, nu, gibbs2[[j]]$U[i,])
    # Alpha
    gibbs2[[j]]$alpha[i] <- draw.alpha(n, gibbs2[[j]]$mu[i], gibbs2[[j]]$U[i,])
    # Sigma
    gibbs2[[j]]$sigma[i] <- gibbs2[[j]]$alpha[i] * gibbs2[[j]]$tau[i]
  }
```

}

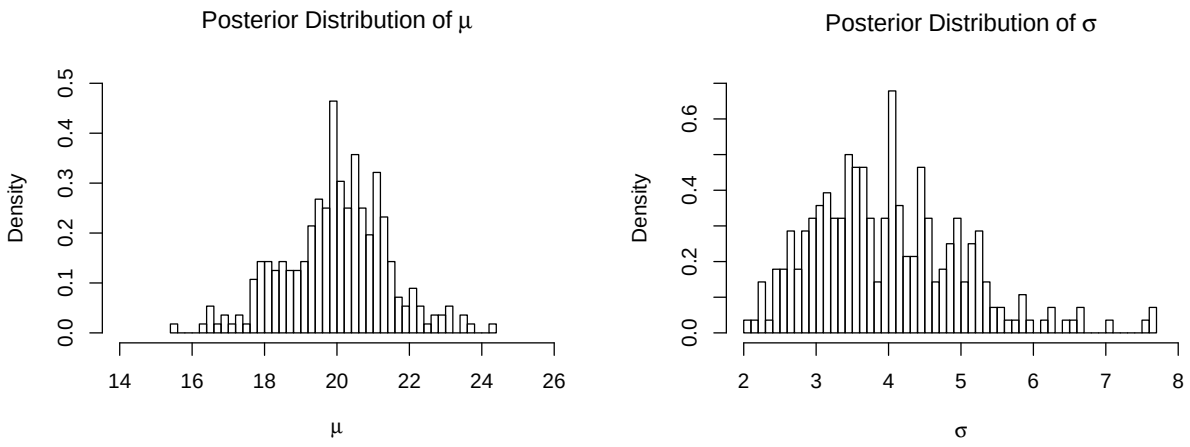

Figure 3: Posterior distributions from the parameter expansion model.

(d) Figure 4 shows the first 100 simulations of μ and σ from both models. In both, μ converged quickly and exhibited little autocorrelation.

The augmented model from part (b) took about 40 iterations before the draws of σ were fully mixed, and the chains show clear autocorrelation after that. I did not observe σ getting stuck near 0, but the chains had a tendency to wander when σ got large. In contrast, the σ draws from the expanded model converged in under 10 iterations and appear independent.

Traceplots of σ, τ, and α from the second model appear in Figure 5. The dependence between between τ and α is obvious since large values of τ occur with small values of α and vice versa. It seems remarkable that $\sigma = \alpha\tau$ is so well behaved. I imagine it takes practice to see when adding parameters is useful, but it in this case it was easy to implement. I expect I will make good use of reparameterization in the future.

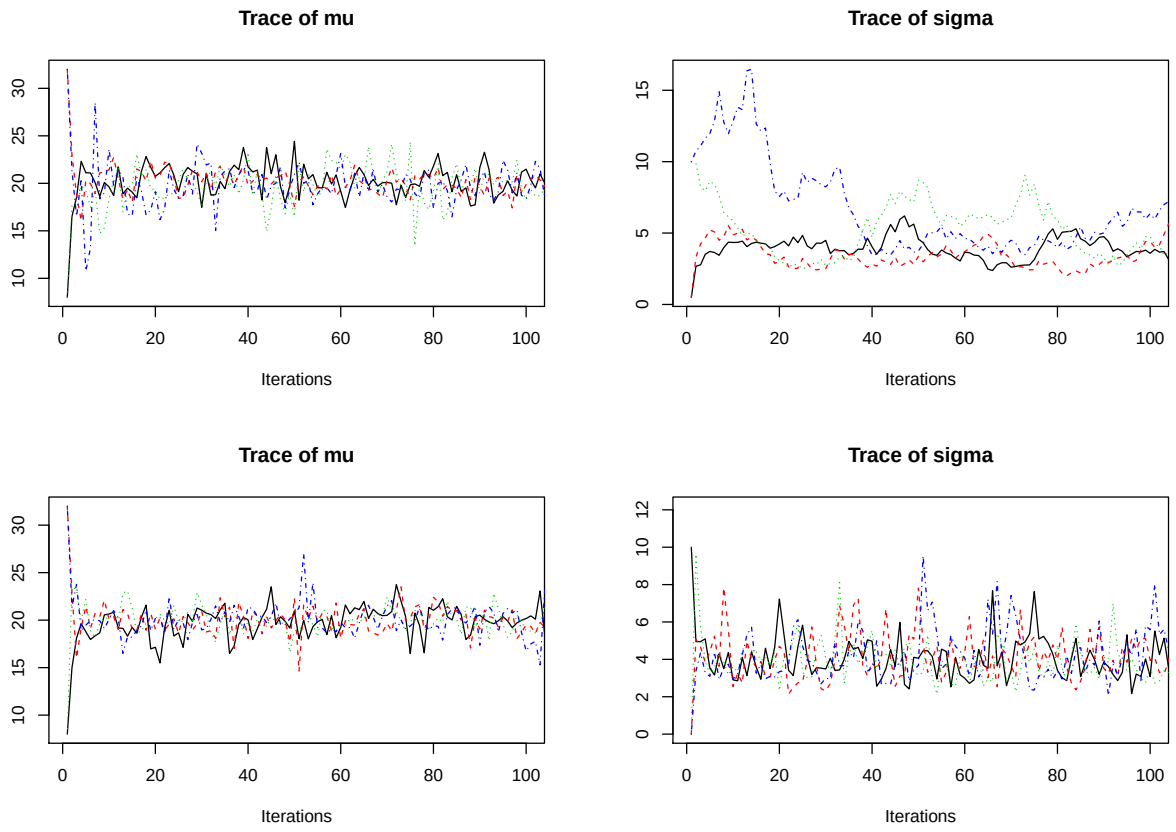

Figure 4: Comparison of traceplots. Top row: The augmented model from part (b). Bottom row: The parameter expansion model from part (c).

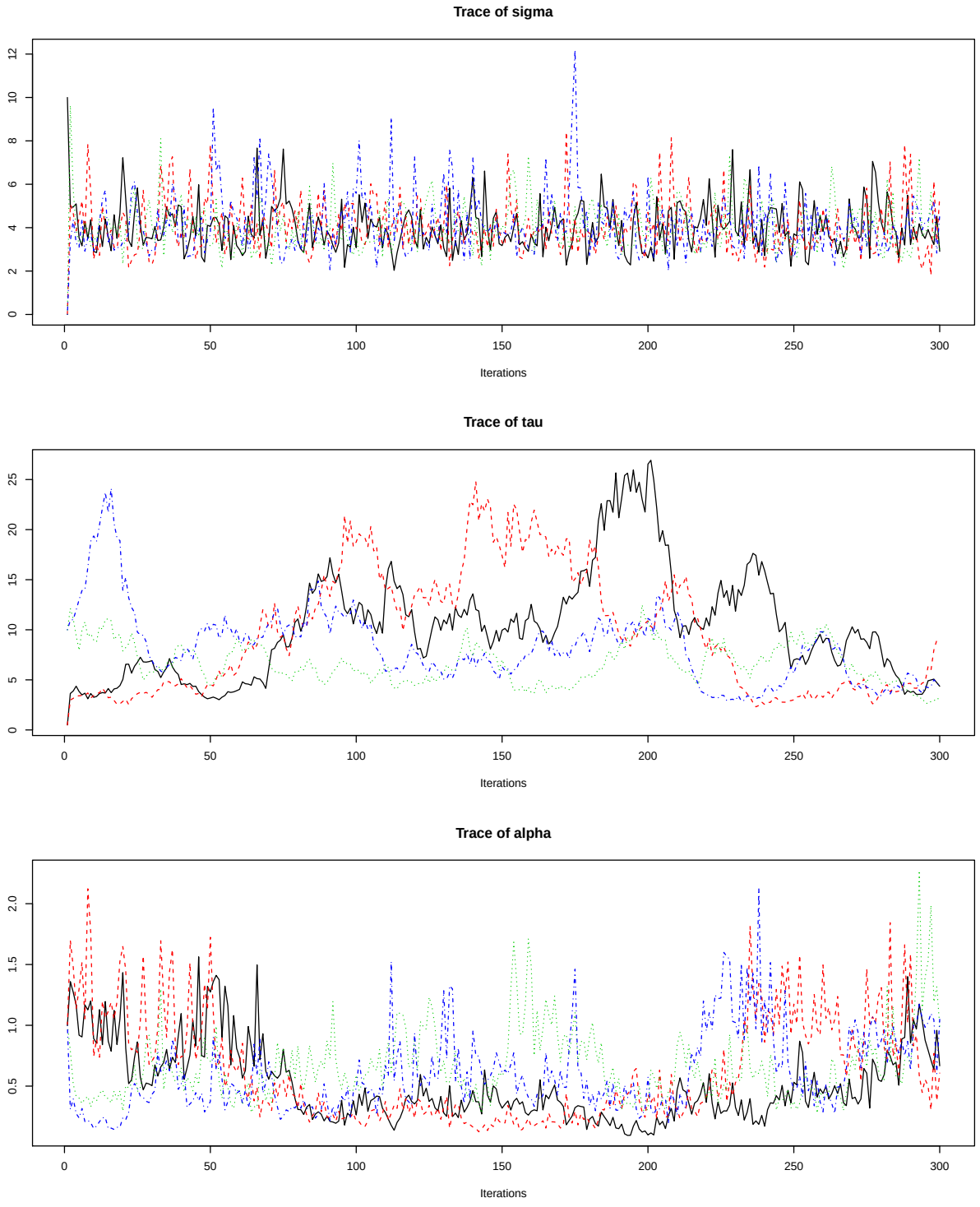

Figure 5: Traceplots for the variance parameters from the augmented model.