Example 6.9: multivariate *t* regression for composition data 1992 House of Commons returns

Simon Jackman

August 16, 2010

Katz and King (1999) examine data from multi-party elections to the United Kingdom's House of Commons. We focus on a subset of the data, covering elections in 1992 in 521 constituencies. The data are *compositional*, in that each observation is a vector of vote proportions that sums to one, by construction: i.e., vote proportions for the Labor Party, the Conservative Party, and the Liberal-Alliance. Votes for all other candidates, spoiled ballots and non-voting are ignored in this analysis, as in the original analysis by Katz and King (1999). In our reanalysis we also ignore seats that were not contested by all three of these parties; we return to this issue below.

A standard approach in the analysis of compositional data is to transform the J-vector of proportions that sum to one,

$$\mathbf{p}_i = (p_{i1}, \ldots, p_{ij})', \; p_{ij} \in [0,1], \forall \, j = 1, \ldots, J; \sum_{j=1}^{j} p_{ij} = 1 \, \forall \, i$$

to a J - 1 vector of log-odds ratios $\mathbf{y}_i$ where $y_{ij} = \log(p_{ij}/p_{ij}) \, \forall \; j = 1, \ldots, J$ - 1; see Aitchison (1986). J = 3 in the analysis here. Katz and King (1999) note that the log-odds ratios generated by these data appear to be heavy-tailed relative to a J - 1-variate normal model, even conditional on covariates, and so use a multivariate t regression model for these data, with unknown degrees of freedom. Katz and King (1999) fit the model via maximum likelihood methods; here we sketch a Bayesian approach, using MCMC methods implemented in `JAGS/BUGS`.

The two log-odds ratios in each $\mathbf{y}_i$ are formed by dividing the Labor and Conservative vote shares by the Liberal/Alliance vote shares. There are seven predictors in the analysis: a constant (x_{i1}), the lagged values (from the previous election) of the two log odds ratios (x_{i2} and x_{i3}), an indicator for whether the Conservative candidate is the incumbent (x_{i4}), the incumbency status of the Labor candidate (x_{i5}), the incumbency status of the Liberal/Alliance candidate (x_{i6}) and an indicator for seats in which no incumbent is running (x_{i7}). Note that $x_{i1} = \sum_{k=4}^{7} x_{ik} = 1 \, \forall \; i$, and so the model is over-parameterized (the underlying regression model suffers from perfect collinearity) and the parameters attaching to those variables are not identified. We proceed ignoring this lack of identification for the moment.

Let $\mathbf{x}_i$ be the vector of covariates. Then the multivariate t regression model is

$$\mathbf{y}_i = \begin{bmatrix} y_{i1} \\ y_{i2} \end{bmatrix} \sim t_\nu \left(\begin{bmatrix} \mathbf{x}_i\boldsymbol{\beta}_1 \\ \mathbf{x}_i\boldsymbol{\beta}_2 \end{bmatrix}, \boldsymbol{\Sigma} \right) \tag{1}$$

where the unknown parameters here are $\boldsymbol{\beta}_j$ are vectors of regression parameters (j = 1, 2), $\boldsymbol{\Sigma}$ is a squared scale matrix (a symmetric, J - 1-by-J - 1, positive definite matrix) and ν is a degrees of freedom parameter (see Appendix B in BASS for the definition of the multivariate t density). Under an assumption of conditional independence given the covariates $\mathbf{x}_i$ and the regression model, the likelihood for this problem is simply the n-fold product of these observation-specific multivariate t densities. The model is closely related to the seemingly-unrelated regression (SUR) model (Zellner 1962) save for the use of the multivariate t density in place of the multivariate normal.

A Bayesian analysis requires a prior for the elements $\boldsymbol{\theta}$. I impose the prior independence assumptions:

$$p(\boldsymbol{\theta}) = p(\sigma_{11})p(\sigma_{12})p(\rho)p(\nu) \times \prod_{k=1}^{7}\prod_{j=1}^{2} p(\beta_{kj})$$

where $\beta_{kj} \sim N(0, 1000)$, $p(\sigma_{jj}) \sim$ Unif(0, .5), $\rho \sim$ Unif(-1, 1), $\sigma_{12} = \rho\sigma_{11}\sigma_{22}$ and $\nu \sim$ Unif(3, 30). Note that we do not impose a inverse-Wishart prior on the squared scale matrix $\mathbf{\Sigma}$, but put uniform priors on the diagonal elements σ_{11} and σ_{22}, and then recover a positive definite $\mathbf{\Sigma}$ by defining $\sigma_{12} = \rho\sigma_{11}\sigma_{22}$ where $\rho \sim$ Unif(-1, 1).

The uniform prior on the degrees of freedom parameter ν encompasses a very heavy-tailed density at the lower bound of $\nu = 3$; although the t tends to the normal as $\nu \to \infty$, a t density with degrees of freedom at the upper bound of $\nu = 30$ is virtually indistinguishable from the normal. In this sense the prior on ν is quite permissive, allowing the data to adjudicate between the normal or a low degree-of-freedom, heavy-tailed t density. Note also that while we are used to thinking of degrees of freedom as positive integers, in the context of indexing a t density *per se* the restriction to integer values of ν is unnecessary.

Speeding up the MCMC algorithm via over-parameterization. As noted above, the model is over-parameterized. We do this deliberately. A lack of identification poses no formal problem for Bayesian analysis: so long as the prior density is proper, so too is the posterior density. And more pragmatically, a MCMC algorithm that explores a posterior density over a set of unidentified parameters often generates an efficient exploration of the posterior density defined over the (lower-dimensional) space of identified parameters. That is, in many sitautions, it is to our advantage to specify a model that contains unidentified parameters, let the MCMC algorithm run, and then examine the output of the algorithm projected down into the space of identified parameters. This is a strategy we will explore in greater detail in the context of hierarchical models.

For now, note that the mapping from the set of unidentified parameters to identified parameters is trivial. The unidentified model includes an intercept plus a set of four mutually exclusive and exhaustive indicators of election type. Identification can obtained by redefining the coefficients on the election type indicators (β_{4j} through β_{7j}, $j = 1, 2$) as offsets around the intercept β_{1j}. This is accomplished by imposing the constraint $\sum_{k=4}^{7} \beta_{kj} = 0, j = 1, 2$. We implement this by letting the MCMC algorithm run with the unidentified model. At each iteration t we compute $\beta_{kj}^{*(t)} = \beta_{kj}^{(t)} - \bar{\beta}_{j}^{(t)}, k = 4, \ldots, 7$ and $j = 1, 2$, and where $\bar{\beta}_{j}^{(t)} = (1/4)\sum_{k=4}^{7} \beta_{kj}^{(t)}, j = 1, 2$. In this way we have imposed the constraint $\sum_{k=4}^{7} \beta_{kj}^{*(t)} = 0, j = 1, 2, \forall\ t$, with β_{kj}^{*} being the set of identified parameters. We can implement this mapping from unidentified to identified parameters either in the JAGS/BUGS program itself, or "post-process" the MCMC output in R. In this case we do the latter.

Implementation. We begin by reading the data we got from King and Katz.

R Code

```
> data <- dget(file="92covs.r.dpt")
> x <- t(data$x)
> y <- t(data$y)
> dim(x)
```

```
[1] 521   6
```

The **X** matrix has 6 columns. The last three columns are binary indicators for the incumbency status of the seat:

R Code

```
> apply(x,2,summary)
```

```
        [,1]    [,2]    [,3]   [,4]   [,5]    [,6]
Min.       1 -1.3280 -2.0140 0.0000 0.0000 0.00000
1st Qu.    1  0.4638 -0.5540 0.0000 0.0000 0.00000
Median     1  0.7302  0.2619 1.0000 0.0000 0.00000
Mean       1  0.6748  0.1526 0.5739 0.2668 0.01536
3rd Qu.    1  0.9462  0.9864 1.0000 1.0000 0.00000
Max.       1  1.6060  1.8750 1.0000 1.0000 1.00000
```

We create a 7-th column for open seats as follows:

R Code

```
> openSeat <- as.numeric(x[,4]!=1 & x[,5]!=1 & x[,6]!=1)
> x <- cbind(x,openSeat)
```

At this point, some informative variable names might be helpful:

R Code

```
> dimnames(x) <- list(NULL,
+                     c("Intercept",
```

```
+                       "lagLogOdds1",
+                       "lagLogOdds2",
+                       "incCons",
+                       "incLab",
+                       "incLibDem",
+                       "openSeat"))
```

We now dump the data in a form that can go into my R package, **pscl**:

R Code

```
> UKHouseOfCommons <- list(x=x,y=y)
> save("UKHouseOfCommons",
+      file="UKHouseOfCommons.rda")
```

The JAGS model is as follows:

JAGS code

```
model{
        for(i in 1:521){
              for(j in 1:2){
                        mu[i,j] <- inprod(x[i,1:7],beta[1:7,j])
                }
                y[i,1:2] ~ dmt(mu[i,1:2],Prec,nu); ## multivariate t
        }

        ## priors
        for(j in 1:2){
              beta[1:7,j] ~ dmnorm(b0,B0) ## 2 vectors of regression parameters
              s[j] ~ dunif(0,.5)          ## sqrt root of diagonal of Sigma
        }
        rho ~ dunif(-1,1)                 ## auxiliary parameter

        ## form Sigma
        Sigma[1,1] <- pow(s[1],2)
        Sigma[2,2] <- pow(s[2],2)
        Sigma[1,2] <- rho*s[1]*s[2]       ## off-diagonal of pos-def Sigma
        Sigma[2,1] <- Sigma[1,2]
        Prec <- inverse(Sigma)            ## convert to precision matrix

        nu ~ dunif(3,30)                  ## degrees of freedom parameter
}
```

This code makes use of the dmt construct, denoting a multivariate *t* density; a univariate *t* density uses the dt construct in JAGS/BUGS. Like many densities in BUGS this function takes the inverse squared scale matrix as an argument, rather than squared scale matrix itself (i.e., recall that the dnorm and dmnorm functions take precisions as their second arguments). Note also that the unknown degrees of freedom parameter nu appears as the third argument to the dmt construct.

We also pass the covariates from R to JAGS as a matrix **X**, and form the linear predictor $\mu_{ij} = \mathbf{x}_i\boldsymbol{\beta}_j$ via the inprod function. As is often the case with specifying models in JAGS and BUGS, the model for the data is easy to specify; the bulk of the program is actually devoted to specifying the priors and making transformations. The hyper-parameters b0 and B0 are passed to JAGS along with the data y and x.

We now get the data ready for the hand-off to JAGS. Note the two sets of initial values, because we will run two chains in parallel for this problem.

R Code

```
> forJags <- list(y=UKHouseOfCommons$y,
+                 x=UKHouseOfCommons$x,
+                 b0=rep(0,7),
+                 B0=diag(.0001,7))
> inits <- list(list(beta=matrix(0,7,2),    ## 2 sets of initial values
+                    s=runif(n=2,0,.5),
+                    rho=runif(n=1,-1,1),
+                    nu=4,
+                    .RNG.name="base::Mersenne-Twister",
+                    .RNG.seed=1234),
+               list(beta=matrix(0,7,2),
+                    s=runif(n=2,0,.5),
+                    rho=runif(n=1,-1,1),
+                    nu=29,
+                    .RNG.name="base::Mersenne-Twister",
+                    .RNG.seed=314159)
+               )
```

Two sets of initial values are passed to JAGS in the object `inits`, a list of lists. Note that we randomly generate start values for `s` and `rho`. This helps ensure that the chains start at different parts of the parameter space, as do the different start values for `nu`.

We now attempt to compile the model in JAGS. Note the longer than usual adaption phase; the default of 1,000 adaption iterations produced a warning message.

R Code

```
> library(rjags)
> foo <- jags.model(file="92plain.bug",
+                   data=forJags,
+                   inits=inits,
+                   n.chains=2,
+                   n.adapt=10e3)
```

```
module basemod loaded
module bugs loaded

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 6328
```

We request 50,000 iterations:

R Code

```
> out <- coda.samples(foo,
+                     variable.names=c("beta","Sigma","nu"),
+                     n.iter=50e3)
```

After the JAGS program has run we map the output of the two chains on the space of unidentified parameters into the space of identified parameters, via the transformations discussed above. The transformations are not at all difficult to implement (row-wise, mean differencing of a matrix) and the bulk of the work is keeping input and output aligned, and dealing with the fact that we have J - 1 sets of regression coefficients to remap. We implement this with the following R code, with the work being done inside a function, `remap`.

R Code

```
> ## map into space of identified parameters
> remap <- function(x){
+   foo <- x                          ## copy MCMC output
+   int <- x[,"beta[1,1]"]
+   k <- cbind(x[,"beta[4,1]"],       ## subset of parameters we need
+              x[,"beta[5,1]"],
+              x[,"beta[6,1]"],
+              x[,"beta[7,1]"])
+   kbar <- apply(k,1,mean)           ## mean at each iteration
+   k <- sweep(k,1,kbar)              ## subtract out
+   foo[,"beta[1,1]"]  <- int + kbar
+   foo[,"beta[4,1]"] <- k[,1]        ## write mean-differenced to output
+   foo[,"beta[5,1]"] <- k[,2]
+   foo[,"beta[6,1]"] <- k[,3]
+   foo[,"beta[7,1]"] <- k[,4]
+
+   int <- x[,"beta[1,2]"]
+   k <- cbind(x[,"beta[4,2]"],       ## repeat for 2nd set of regression coefficients
+              x[,"beta[5,2]"],
+              x[,"beta[6,2]"],
+              x[,"beta[7,2]"])
+   kbar <- apply(k,1,mean)
+   k <- sweep(k,1,kbar)
+   foo[,"beta[1,2]"] <- int + kbar
+   foo[,"beta[4,2]"] <- k[,1]
+   foo[,"beta[5,2]"] <- k[,2]
+   foo[,"beta[6,2]"] <- k[,3]
+   foo[,"beta[7,2]"] <- k[,4]
+
+   return(foo)
+ }
> z <- lapply(out,remap)              ## apply to each chain in out
> ## drop duplicate Sigma[2,1]==Sigma[1,2]
> z <- lapply(z,
+             function(x)x[,-grep("Sigma[2,1]",dimnames(x)[[2]],fixed=TRUE)])
```

We now execute code producing some customized trace plots for these data:

R Code

```
> mytrace <- function(obj,var){
+   z <- cbind(obj[[1]][,var],
+              obj[[2]][,var])
+   plot.ts(z,
+           plot.type="single",
+           col=c("black",gray(.45)),
+           lwd=.1,
+           axes=FALSE,
+           xlab="",
+           ylab="")
+   axis(1)
+   axis(2)
+   mtext(at=.98*par()$usr[1]+.02*par()$usr[2],
+         side=3,
+         line=-1,
+         adj=0,
+         cex=.75,
+         as.expression(parse(text=sub(var,pattern=",",replacement=""))))
+   invisible(NULL)
+ }
> theOnes <- c("beta[1,1]","beta[4,1]","beta[5,1]","beta[6,1]","beta[7,1]")
> pdf(file="traces.pdf",
+     width=6.5,
+     height=7.5)
> par(mfrow=c(5,2),
+     mar=c(2,2,1,1),
+     cex.axis=.75,
+     cex.lab=.85,
+     tcl=-.2,
+     las=1,
+     oma=c(0,0,1.5,0),
+     mgp=c(1.65,.3,0))
> for(i in theOnes){
+   mytrace(out,i)
+   mytrace(z,i)
+ }
> print(par()$usr)
```

```
[1] -1.998960e+03  5.199996e+04 -8.560217e-02  1.869679e-01
```

R Code

```
> mtext(outer=TRUE,side=3,"Unidentified",at=.25)
> mtext(outer=TRUE,side=3,"Identified",at=.75)
> dev.off()
```

```
null device
          1
```

Figure 1 shows trace plots for the unidentified parameters (left panel) and the transformed, identified parameters in the right panel.

We also produce graphical summaries of the performance of the algorithm with respect to the degrees of freedom parameter, ν, with the following code. The resulting graph appears below as Figure 2.

R Code

```
> pdf(file="nu.pdf",
+     colormodel="gray",
+     w=6.5,
+     h=6.5)
> layout(matrix(c(1,1,2,3),2,2,byrow=TRUE))
> par(mar=c(3,2.5,1,1),
+     las=1,
+     cex.axis=.75,
+     cex.lab=.85,
+     mgp=c(1.5,.6,0))
> nu <- cbind(out[[1]][,"nu"],
+             out[[2]][,"nu"])
> nu <- mcmc(nu[seq(1,50e3,length=5e3),])
> plot.ts(nu,
+         axes=FALSE,
+         xlab="Iteration",
+         ylab=expression(nu),
+         plot.type="single",
+         lwd=.1,
+         col=c("black",gray(.45)))
> xTics <- c(0,seq(1000,5000,by=1000))
```

```
> axis(1,at=xTics,labels=xTics*10)
> axis(2)
> a <- apply(cbind(out[[1]][,"nu"],
+                  out[[2]][,"nu"]),
+            2,acf,plot=FALSE)
> plot.ts(cbind(a[[1]]$acf[,,1],
+               a[[2]]$acf[,,1]),
+         lwd=2,
+         plot.type="single",
+         ylab="Autocorrelations",
+         col=c("black",
+           rgb(115/255,115/255,115/255,1)))
> d <- lapply(out,function(x)density(x[,"nu"],from=3,to=30,cut=0))
> yMax <- max(unlist(lapply(d,function(x)max(x$y))))
> plot(d[[1]],
+      ylim=c(0,yMax),
+      lwd=3,
+      xlim=c(3,10),
+      type="l",
+      xlab=expression(nu),
+      ylab="",
+      main="",
+      axes=FALSE)
> lines(d[[2]]$x,d[[2]]$y,
+       lwd=3,
+       col=rgb(115/255,115/255,115/255,1))
> abline(h=1/27,lty=2)   ## overlay prior
> axis(1)
> dev.off()
```

```
null device
          1
```

The identified parameters mix reasonably well, with Gelman and Rubin R statistics generally no larger than 1.03:

R Code

```
> ## gelman-rubin
> k <- dim(z[[2]])[2]
> gr <- rep(NA,k)
> for(j in 1:k){
+   gr[j] <- gelman.diag(mcmc.list(z[[1]][,j],z[[2]][,j]))$psrf[1,1]
+ }
> print(gr)
```

```
 [1] 1.000202 1.000132 1.000121 1.015353 1.002876 1.001544 1.022165 1.001990
 [9] 1.019394 1.030165 1.011473 1.005645 1.001618 1.016523 1.001603 1.017466
[17] 1.028072 1.000185
```

On the other hand, Raftery-Lewis diagnostics suggest that the several hundred thousand iterations may be required to generate accurate estimates of the extreme quantiles of some the marginal posterior densities of $\boldsymbol{\beta}_j$; JAGS uses a Metropolis sampler to update these parameters and may be suffering from a poor choice of a proposal density:

R Code

```
> raftery.diag(mcmc.list(z[[1]],z[[2]]))
```

```
[[1]]

Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

            Burn-in  Total  Lower bound  Dependence
             (M)      (N)    (Nmin)       factor (I)
 Sigma[1,1] 16       21444  3746          5.72
 Sigma[1,2] 15       18291  3746          4.88
 Sigma[2,2] 16       20952  3746          5.59
 beta[1,1]  476      525322 3746        140.00
 beta[2,1]  141      152184 3746         40.60
 beta[3,1]  165      181320 3746         48.40
 beta[4,1]  648      680832 3746        182.00
 beta[5,1]  256      280496 3746         74.90
 beta[6,1]  556      605454 3746        162.00
```

```
 beta[7,1]  250      250100 3746          66.80
 beta[1,2]  207      224703 3746          60.00
 beta[2,2]  245      257698 3746          68.80
 beta[3,2]  180      198414 3746          53.00
 beta[4,2]  294      318255 3746          85.00
 beta[5,2]  260      275500 3746          73.50
 beta[6,2]  230      250872 3746          67.00
 beta[7,2]  126      135104 3746          36.10
 nu         6        9046   3746           2.41


[[2]]

Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

            Burn-in  Total  Lower bound  Dependence
            (M)      (N)    (Nmin)       factor (I)
 Sigma[1,1] 15       18561  3746           4.95
 Sigma[1,2] 15       17511  3746           4.67
 Sigma[2,2] 16       19880  3746           5.31
 beta[1,1]  357      409377 3746         109.00
 beta[2,1]  238      260659 3746          69.60
 beta[3,1]  170      169116 3746          45.10
 beta[4,1]  133      144548 3746          38.60
 beta[5,1]  171      184365 3746          49.20
 beta[6,1]  218      237098 3746          63.30
 beta[7,1]  123      133271 3746          35.60
 beta[1,2]  105      113586 3746          30.30
 beta[2,2]  209      243012 3746          64.90
 beta[3,2]  176      195936 3746          52.30
 beta[4,2]  109      118467 3746          31.60
 beta[5,2]  129      140354 3746          37.50
 beta[6,2]  138      149580 3746          39.90
 beta[7,2]  90       98096  3746          26.20
 nu         6        11946  3746           3.19
```

References

Aitchison, J. 1986. *The Statistical Analysis of Compositional Data*. London: Chapman and Hall.

Katz, Jonathan N. and Gary King. 1999. "A Statistical Model for Multiparty Electoral Data." *American Political Science Review* 93:15--32.

Zellner, Arnold. 1962. "An Efficient Method of Estimating Seemingly Unrelated Regressions and Tests for Aggregation Bias." *Journal of the American Statistical Association* 57:348--368.

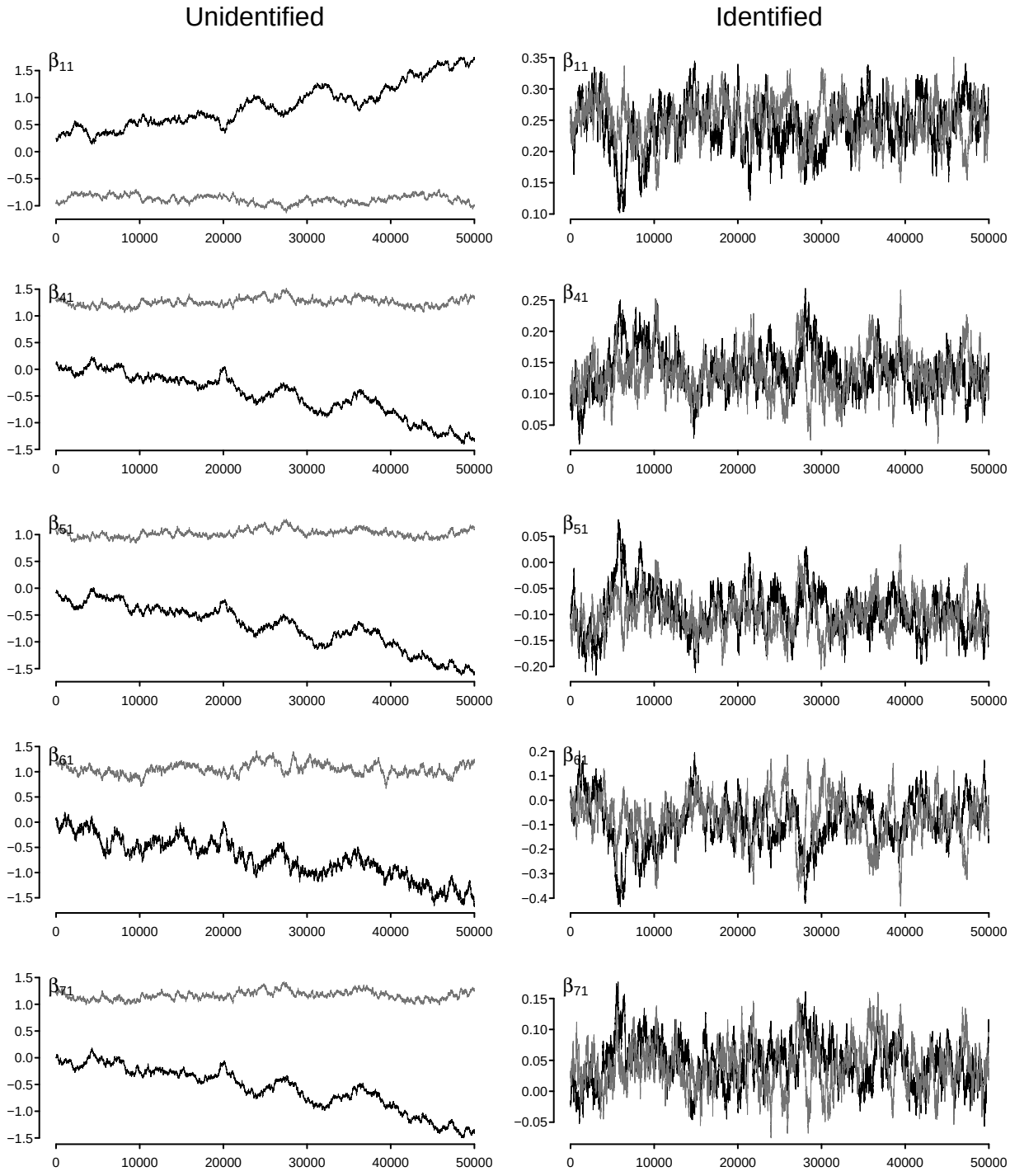

Figure 1: Trace plots, unidentified and identified parameters. Each panel shows trace plots from the two chains; panels on the right are for identified panels. Note the similarity of the trace plots within chains, across parameters, for the unidentified parameters.

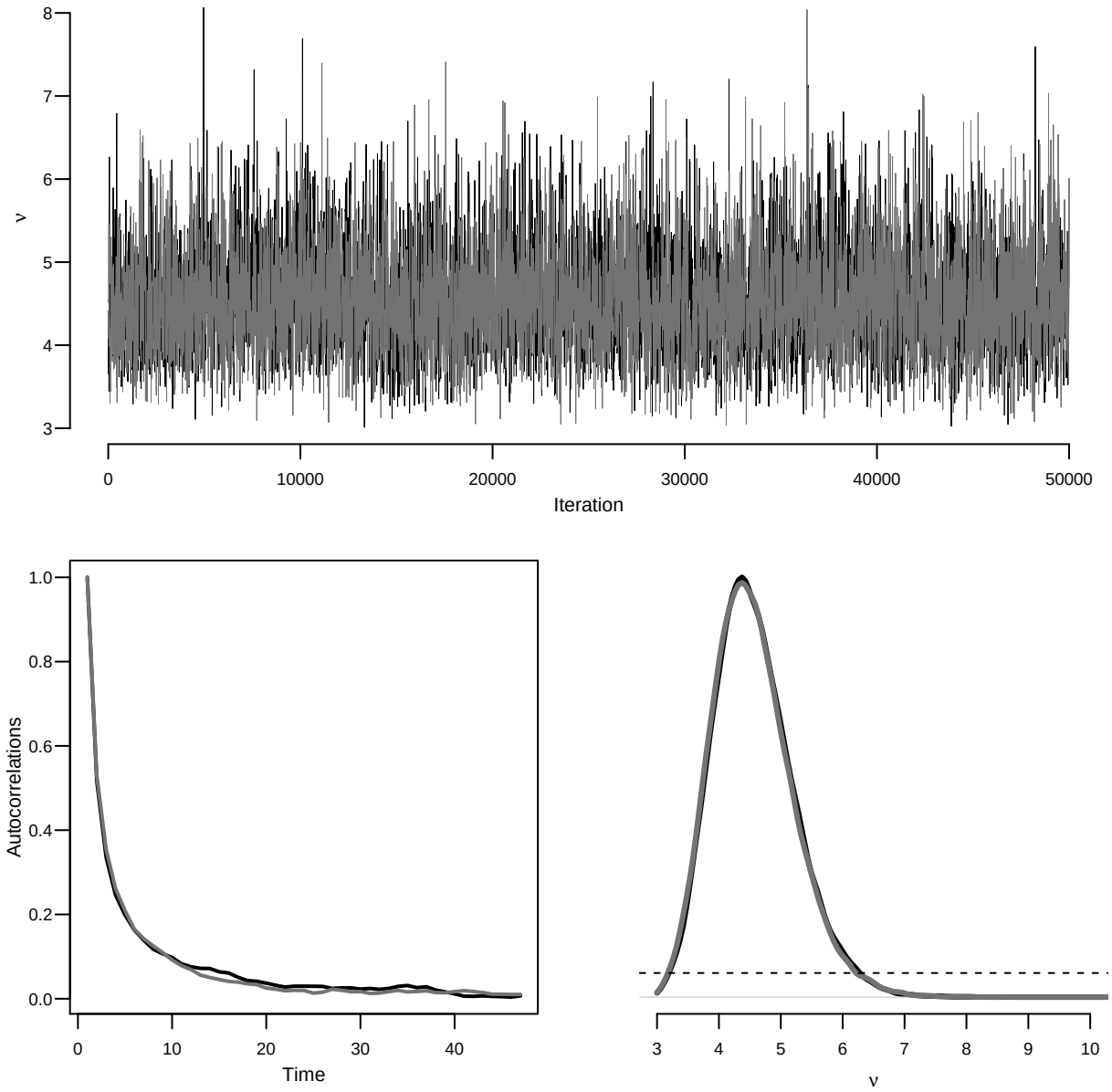

Figure 2: Traceplots, autocorrelation spectra and estimated posterior density for ν, the degrees of freedom parameter.