Household Radon Measurements in Minnesota: An Example of Bayesian Statistics in Geosciences[*]

Jonathan Gilligan

Jul. 18, 2014

Abstract

This document summarizes some statistical analysis of the spatial distribution of the concentration of radon gas in homes in the US (with a focus on Minnesota) that Phil Price and Andrew Gelman conducted in the early 1990s (Price, Nero & Gelman 1996), and which became a thread in two textbooks Gelman wrote (Gelman & Hill 2006; Gelman *et al.* 2013).

It covers simple multilevel modeling with an emphasis on using Bayesian analysis to estimate parameters, provides examples of using JAGS to perform Bayesian analysis, and discusses diagnosing JAGS output to check whether there are signs of problems.

Contents

[*]Adapted from Gelman & Hill 2006

List of Figures

1 Introduction: Measuring Radon

Radon is a radioactive gas that is released by the decay of uranium and other radioactive elements in naturally occurring minerals in many parts of the United States. When radon seeps into homes and accumulates in high concentrations, it can cause lung cancer, and it is estimated to cause several thousand deaths per year in the U.S. There is considerable variation in the amount of radon in U.S. houses, due both to variations in home construction and the amount of radon-producing minerals in the ground. In the 1990s, the Environmental Protection Agency conducted a survey in which indoor radon concentrations were measured in a sample of more than 80,000 randomly selected houses. However, 80,000 houses is a very small sample of the more than 75 million single-family houses in the US. If we think about this at the county-level, there are 3000 counties, so on average almost 30 houses were sampled per county, but because of the random selection of the sample and the uneven distribution of housing across counties, some counties ended up represented by only a small number of measurements. Therefore, sophisticated statistical analyses was necessary to estimate the radon hazard in all 3,000 US counties.

Andrew Gelman, in collaboration with physicist Phil Price (1996), took a Bayesian approach, as Price describes here (Price 2014):

> *My first encounter with Bayesian statistics was just over 20 years ago. I was a postdoc at Lawrence Berkeley National Laboratory, with a new PhD in theoretical atomic physics but working on various problems related to the geographical and statistical distribution of indoor radon (a naturally occurring radioactive gas that can be dangerous if present at high concentrations). One of the issues I ran into right at the start was that many researchers were trying to use data from statewide radon surveys to make maps of radon concentrations within states. The surveys had been designed to characterize the statistical distribution of radon in each state, not the spatial distribution, so sample sizes were much higher in highly populated counties than in low-population counties.*
>
> *Within the counties with lots of measurements, the statistical distribution of radon measurements was roughly lognormal, with a geometric standard deviation of around 3 (a dimensionless number) and a geometric mean that varied from county to county. One of the first datasets I looked at in detail was from Minnesota, where counties with more than 100 radon survey measurements had observed geometric means as low as 83 Becquerels per cubic meter and as high as 136 Becquerels per cubic meter.*
>
> *100 radon measurements from a county is enough to characterize the county's radon concentration pretty well, especially if one is willing to assume the statistical distribution of measurements is lognormal; for instance, one of the parameters of interest to policy-makers is the expected fraction of homes with radon concentrations exceeding the recommended "action level" of 150 Becquerels per cubic meter, and with 100 measurements you could estimate that fraction to within a few percentage points, which was close enough.*
>
> *But what about counties with many fewer measurements? Many counties had a number of measurements in the single digits. Unsurprisingly, by far the lowest and highest observed geometric mean radon concentrations were from counties with very few measurements: if you only measure in two homes in a county, 4% of the time they will both come from the highest 20% of homes in the county so your geometric mean will be much higher than the true geometric mean, and 4% of the time they will both come from the lowest 20% of homes in the county, with the opposite result. I still remember, 20 years later, that the highest county geometric mean was in Lac Qui Parle County, with just two sampled homes and an observed geometric mean of 500 Becquerels per cubic meter.*

As Gelman describes the goal of this analysis (Gelman & Hill 2006, p. 3),

> *Our goal in analyzing these data was to estimate the distribution of radon levels in each of the approximately 3000 counties in the United States, so that homeowners could make decisions about measuring or remediating the radon in their houses based on the best available knowledge of local conditions. For the purpose of this analysis, the data were structured hierarchically: houses within*

counties. . . . In performing the analysis, we had an important predictor—the floor on which the measurement was taken, either basement or first floor; radon comes from underground and can enter more easily when a house is built into the ground. We also had an important county-level predictor—a measurement of soil uranium that was available at the county level.

2 Multilevel Modeling

Start out our R session by loading the libraries we will need.

```
if (!require(rjags))   stop("Could not load rjags")
if (!require(lme4))    stop("Could not load lme4")
if (!require(ggplot2)) stop("Could not load ggplot2")
if (!require(ggmcmc))  stop("Could not load ggmcmc")
if (!require(stringr)) stop("Could not load stringr")
if (!require(xtable))  stop("Could not load xtable")
if (!require(BEST))    stop("Could not load BEST")
```

Now, we read in the data:

```
srrs2 <{ read.table ("srrs2.dat", header=T, sep=",")
mn <{ srrs2$state=="MN"
#
# activity is concentration of radon in  picocuries per liter (pCi/l).
#
radon <{ srrs2$activity[mn]
```

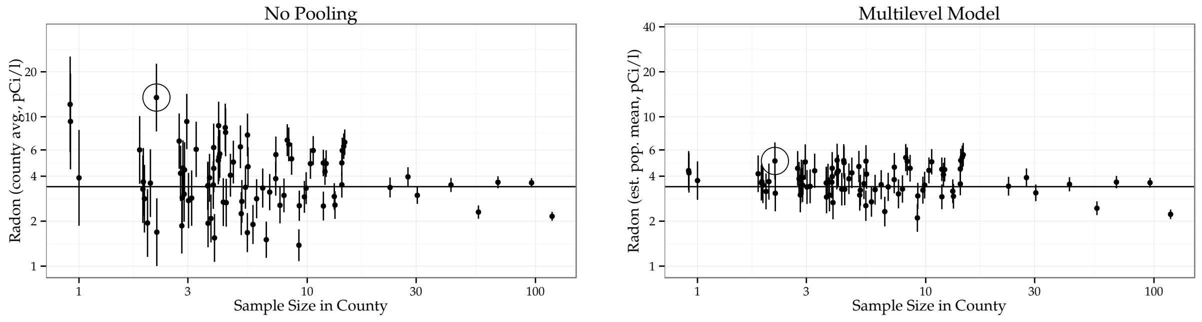
Gelman comments that "Radon levels are always positive, and it is reasonable to suppose that effects will be multiplicative; hence it is appropriate to model the data on the logarithmic scale." Moreover, as Price notes in the quotation above, the radon concentration is approximately log-normally distributed in each county, so this further reinforces the notion that we will do well to work with the logarithm of the distribution (which will then be normally distributed within each county). However, there are measurements that are zero, so the logarithm would be $-\infty$. To deal with this, we assign those values a small positive number. This is a fudge, but it makes the analysis go more smoothly and does not substantively affect the outcome.

Another complication is that some houses have basements and others do not. The radon measurements are always performed on the lowest level of a house, and basements will generally have much higher radon concentrations, both because it is easier for radon to diffuse from the soil through the basement walls with basements and because radon is heavier than normal air, so it tends to pool in low places. Thus, we expect radon measurements to be higher in houses with basements than in houses built above ground on a slab.

```
log.radon <{ log (ifelse (radon==0, 0.1, radon))
x <{ srrs2$floor[mn]       # 0 for basement, 1 for ground floor
n <{ length(radon)
y <{ log.radon

# get county index variable
county.name <{ as.vector(srrs2$county[mn])
uniq <{ unique(county.name)
n.counties <{ length(uniq)
lqp.index <{ which(str_trim(uniq) == 'LAC QUI PARLE')
county <{ rep (NA, n.counties)
for (i in 1:n.counties){
  county[county.name==uniq[i]] <{ i

  sample.size <{ as.vector (table (county))
}
```

2.1 Pooling Data

Now that we have loaded the data, we can plot it and calculate a few simple descriptive statistics to get a sense of it. This analysis follows Gelman & Hill 2006, section 12.2, and I have copied most



(a) *No-pooling county-level analysis.* **(b)** *Multilevel (partial pooling) analysis.*

Figure 1: *Estimates ± standard errors for the average log radon levels in Minnesota counties plotted versus the (jittered) number of measurements in the county. In both analyses, no county-level or house-level effects are considered. The counties with fewer measurements have more variable estimates with higher standard errors. The horizontal line in each figure shows an estimate of state-wide average (complete pooling of all counties). Fig. 1a illustrates a problem with the no-pooling analysis: it systematically causes us to think that certain counties are more extreme, just because they have smaller sample sizes. Lac Qui Parle county, which has the highest measured radon level, but only 2 measurements, is circled.*

of the R code and the design of the figures verbatim. There are 85 counties, and the number of measurements per county ranges from 1 to 116. There are two extremes of how we can characterize these measurements: We can pool all the counties and compute the average amount of radon for all of Minnesota, which will give us a very reliable measurement of the state average, but will ignore any variation from one county to another. This would be pretty useless for advising homeowners in different parts of the state whether to test their homes for radon. On the other hand, we can avoid pooling altogether and report the average for each county individually. This will show us a lot about the spatial variation in radon concentration from one county to another, and will give good accuracy for counties with a lot of measurements, but for counties with only a few measurements the estimate may be too uncertain to provide useful guidance to homeowners in that county. Fig. 1a shows both the completely pooled state average and the completely unpooled county averages.

```
# complete pooling: state average of log(radon)
ybarbar <{ mean(y)

# jitter data in x direction so different counties
# with the same number of measurements don't overlap
sample.size.jittered <{ sample.size*exp (runif (n.counties, -.1, .1))
cty.mns <{ tapply(y,county,mean)
cty.vars <{ tapply(y,county,var)
cty.sds <{ mean(sqrt(cty.vars[!is.na(cty.vars)]))/sqrt(sample.size)
cty.sds.sep <{ sqrt(tapply(y,county,var)/sample.size)

y.limits <{ c(1,35)
```

```
## Figure 1(a)
frame1a <{ data.frame( x = sample.size.jittered, y = exp(cty.mns),
                      y.min = exp(cty.mns - cty.sds),
                      y.max = exp(cty.mns + cty.sds) )
frame1a.lqp <{ data.frame( x = frame1a$x[lqp.index], y = frame1a$y[lqp.index] )
p1a <{ ggplot(frame1a, aes(x = x, y = y)) +
  geom_point(aes(x = x, y = y), data=frame1a.lqp, shape = I(1), size = I(10)) +
  scale_y_log10("Radon (county avg., pCi/l)",
                limits = y.limits,
                breaks=c(1,2,4,6,10, 20)) +
  scale_x_log10("Sample Size in County", breaks=c(1,3,10,30,100,300)) +
  theme_bw() +
  geom_pointrange(aes(ymin = y.min, ymax = y.max, y=y)) +
  geom_hline(yintercept=exp(ybarbar)) +
  labs(title="No Pooling")
print(p1a)
```

2.2 Partial pooling estimates from a multilevel model

Gelman and Hill write (p. 253), ‘‘the no-pooling model overstate the variation among the counties and tend to make the individual counties look more different [from one another] than they actually are.’’ We can see this particularly with Lac Qui Parle county (circled in Fig. 1a), which has the highest average radon concentration of the 85 counties. But there are only 2 measurements for Lac Qui Parle county, so our estimate of the average is very uncertain. Gelman and Hill write, ‘‘Lac Qui Parle county may very well be a high-radon county, but do we really believe it is *that* high?’’

The two extremes of pooling are both unsatisfactory: Complete pooling ignores the variation among counties, and no pooling overstates that variation. Alternately, we can say that analysis with complete pooling under-fits the county-to-county variation and analysis with no pooling over-fits it. Is there an optimal compromise between too much pooling and too little? Price and Gelman used a multilevel model: if α_j represents the actual average log of radon concentration for county j, which we are trying to estimate, then for each county we estimate α_j by a weighted average between the completely-pooled and unpooled values, where the weighting coefficients depend on the standard error for our unpooled estimate in that county:

$$\hat{\alpha}_j^{\text{multilevel}} \approx \frac{(n_j/\sigma_y^2)y_j + (1/\sigma_\alpha^2)y_{\text{all}}}{(n_j/\sigma_y^2) + (1/\sigma_\alpha^2)}, \tag{1}$$

where n_j is the number of measurements in county j, σ_y^2 is the within-county variance in the log radon measurements, and σ_α^2 is the between-county variance. For this model, we assume that the within-county variance is the same for all counties, but we could just as easily allow for varying within-county variances, in which case we would specify $\sigma_{y_j}^2$ in Equation (1).

The trick is that to apply this model to get $\hat{\alpha}_j$, we need to estimate the within-county and between-county variances, but we need α_j to estimate the variances, so it might seem that we are stuck. However, there are two approaches we can use: We can use a linear mixed-effects regression (LMER), which will produce an approximation of the α's and σ's, or we can use a Bayesian approach.

The code below shows the LMER approach, as implemented in R:

```
M0 <{ lmer(y   1 + (1 | county))
summary(M0)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: y ~ 1 + (1 | county)

REML criterion at convergence: 2259

Scaled residuals:
   Min      1Q Median      3Q     Max
-4.466 -0.573  0.044  0.643  3.352

Random effects:
 Groups   Name        Variance Std.Dev.
 county   (Intercept) 0.0958   0.310
 Residual             0.6366   0.798
Number of obs: 919, groups:  county, 85

Fixed effects:
            Estimate Std. Error t value
(Intercept)   1.3126     0.0489    26.8
```

```
coef(M0)$county[lqp.index,]
```

```
[1] 1.61
```

The formula in the call to lmer specifies that the model represents a constant mean value (equivalent to ybarbar, the statewide average) with each county having a randomly distributed county-level effect (the difference between the county-mean and the state mean) and the homes in the county are randomly distributed around the county-level mean.

The code below shows a Bayesian approach to calculating the same model, using JAGS:

2.2.1 JAGS model file

The file below is a JAGS model, `radon_multilevel_nopred.jags`:

```
## @knitr radon_multilevel_nopred_jags
# FILE: radon.multilevel.nopred.jags
#
# JAGS  code for multilevel model for radon
#
model {
  for (i in 1:n){
    y[i]   dnorm (a[county[i]], tau.y)
  }
  tau.y <{ pow(sigma.y, -2)
  sigma.y   dunif (0, 100)

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

JAGS models have two kinds of assignments: <− for deterministic assignments (the variable gets a definite value) and ˜ for stochastic assignments, when the variable is drawn randomly from a specified distribution.

- Each of the measurements y_i is drawn from a normal distribution with mean a_j (where j is the county in which measurement i was taken) and variance σ_y^2 (JAGS parameterizes normal distributions in terms of the *precision*, $\tau = 1/\sigma^2$: precision is the reciprocal of the variance).
- σ_y is drawn from a uniform distribution between 0 and 100.
- a_j is drawn from a normal distribution with mean μ_a and variance σ_a^2.
- μ_a is drawn from a uniform distribution between 0 and 0.0001.
- σ_a is drawn from a uniform distribution between 0 and 100.

JAGS has two kinds of variables: **parameters**, which can vary, and **data**, which has fixed values and does not change. It uses the information on what is data and what is parameters to apply Bayes's theorem to the model:

$$\begin{aligned} P(a_j, \sigma_y, \sigma_a | y_j) &= \frac{\prod_j P(y_j | a_j, \sigma_y, \sigma_a) \times P(a_j, \sigma_y, \sigma_a)}{\int \prod_j P(y_j | a_j, \sigma_y, \sigma_a) \times P(a_j, \sigma_y, \sigma_a) d_{a_j} d_{\sigma_y} d_{\sigma_a}} \\ &= \frac{\prod_j \mathrm{normal}(y_j, a_j, \sigma_y) \mathrm{normal}(a_j, \mu_a, \sigma_a)}{\int \prod_j \mathrm{normal}(y_j, a_j, \sigma_y) \mathrm{normal}(a_j, \mu_a, \sigma_a) \mathrm{normal}(\mu_a, 0, 0.0001) d_{a_j} d_{\sigma_y} d_{\sigma_a} d_{\mu_a}}, \end{aligned} \tag{2}$$

where y_j are the data (measured logarithms of radon concentration) and a_j, σ_y, and σ_a are parameters to be determined.

This is a very nasty equation and none of us would want to have to solve it. Fortunately, that is what JAGS is good at doing automatically:

JAGS works by starting with initial values for each parameter, and calculating the total probability of the combination of parameters and data, and then randomly adjusts the values of each parameter, paying attention to whether the new combination of parameters and data has a higher or lower probability than the previous set. This process, known as Gibbs Sampling, allows JAGS to effectively compute the integral of the evidence term for Bayes's theorem and identify what combination of parameters has the highest posterior probability.

2.2.2 Running JAGS model from R

The code below runs the JAGS model:

```
# We need to initialize the parameters for the model.
# Initialize alpha_j and mu randomly from normal(0,1) distribution
# Initialize sigma randomly from a uniform distribution on [0,1]
radon.inits <{ function (chain){
  list (a=rnorm(n.counties), mu.a=rnorm(1),
        sigma.y=runif(1), sigma.a=runif(1)
  )
}
# prepare a list of data to pass to JAGS model
radon.data <{ list (n = n, n.counties = n.counties, y = y, county = county)
# Tell JAGS the names of the parameters it should report back to us
radon.parameters <{ c ("a", "mu.a", "sigma.y", "sigma.a")

# Compile the JAGS model, initializing the data and parameters
mlm.radon.nopred.model <{ jags.model("radon_multilevel_nopred.jags",
                                   data = radon.data,
                                   inits = radon.inits,
                                   n.chains = 3,
                                   n.adapt=1000)

# After warming up, take 2000 random samples.
update(mlm.radon.nopred.model,n.iter=2000)
mlm.radon.nopred <{ coda.samples( mlm.radon.nopred.model,
                                variable.names = radon.parameters,
                                n.iter = 2000)

# Here, we get the data back from JAGS and convert it to a useful form
post.nopred <{ as.matrix(mlm.radon.nopred)
mean.a.nopred <{ rep (NA, n.counties)
sd.a.nopred <{ rep (NA, n.counties)
for (i in 1:n.counties) {
mean.a.nopred[i] <{ mean(post.nopred[ , paste('a[',i,']', sep='')])
sd.a.nopred[i] <{ sd(post.nopred[ , paste('a[',i,']', sep='')])
}
```

Note that in Fig. 1b, the estimated radon concentration in Lac Qui Parle county has dropped from 13.4 pCi/l when it is estimated with only 2 measurements to 5.1 pCi/l when we partially pool it with the other data.

The code to plot Fig. 1b is shown below:

```
## Figure 1(b)
frame1b <{ data.frame( x = sample.size.jittered,
                      y = exp(mean.a.nopred),
                      y.min = exp(mean.a.nopred - sd.a.nopred),
                      y.max = exp(mean.a.nopred + sd.a.nopred) )
frame1b.lqp <{ data.frame( x= frame1b$x[lqp.index],
                          y= frame1b$y[lqp.index])
p1b <{ ggplot( frame1b, aes(x = x, y = y)) +
  geom_point( aes(x = x, y = y), data = frame1b.lqp, shape = I(1),
              size = I(10) ) +
  scale_y_log10("Radon (est. pop. mean, pCi/l)",
                limits = y.limits,
                breaks=c(1,2,4,6,10, 20, 40)) +
  scale_x_log10("Sample Size in County", breaks=c(1,3,10,30,100,300)) +
  geom_pointrange(aes(ymin=y.min, ymax=y.max, y = y)) +
  geom_hline(yintercept=exp(ybarbar)) +
  theme_bw() +
  labs(title="Multilevel Model")
print(p1b)
```

2.3 Multilevel Modeling: Predictors

But there is more to do: Now we can start thinking about the fact that we know that radon concentration is higher in houses with basements, so, returning to our completely pooled and unpooled analysis, we can add a term for the effect of having a basement. In the completely-pooled version, our model would look like this

$$y_i = \alpha + \beta x_i + \epsilon_i, \tag{3}$$

or in the more compact statistical notation,

$$\mathbf{y} \sim \alpha + \beta \mathbf{x} \tag{4}$$

where $x = 0$ if the house has a basement and $x = 1$ if it doesn't, α represents the statewide average radon concentration, and β represents the effect of having a basement.

```
lm.pooled <{ lm(y   x)
summary(lm.pooled)
```

```

Call:
lm(formula = y ~ x)

Residuals:
   Min     1Q Median     3Q    Max
-3.629 -0.538  0.034  0.560  2.549

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.3267     0.0297   44.64   <2e-16 ***
x            -0.6134     0.0728   -8.42   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.823 on 917 degrees of freedom
Multiple R^2:  0.0718,   Adjusted R^2:  0.0708
F-statistic: 70.9 on 1 and 917 DF,  p-value: <2e-16
```

We can also do an unpooled version, where α varies by county. Here, the model uses the county index (an integer-valued variable that goes from 1 to 85, for the 85 counties).

```
lm.unpooled <{ lm(y   x + factor(county) - 1)
```

The -1 in the formula tells R not to compute a baseline constant term. Otherwise, R would use county #1 as the baseline and calculate the terms for all the other counties as differences from county #1. In both regressions, the slope of the "basement" term is the same for all counties.

The code below produces Fig. 2, which shows the results of the pooled and unpooled analysis for 8 counties. The effect of basements is seen as a sloping line, and **the intercept in each county represents the radon level for a house with a basement in that location**. Using the regression lets us draw meaningful inferences about houses with basements from measurements made in houses without basements.

```
# Fig. 2
display8 <{ c (36, 1, 35, 21, 14, 71, 61, 70)  # counties to be displayed
y.range <{ range (y[!is.na(match(county,display8))])

radon.data <{ data.frame(y, x, county)
radon8.data <{ subset(radon.data, county %in% display8)
```

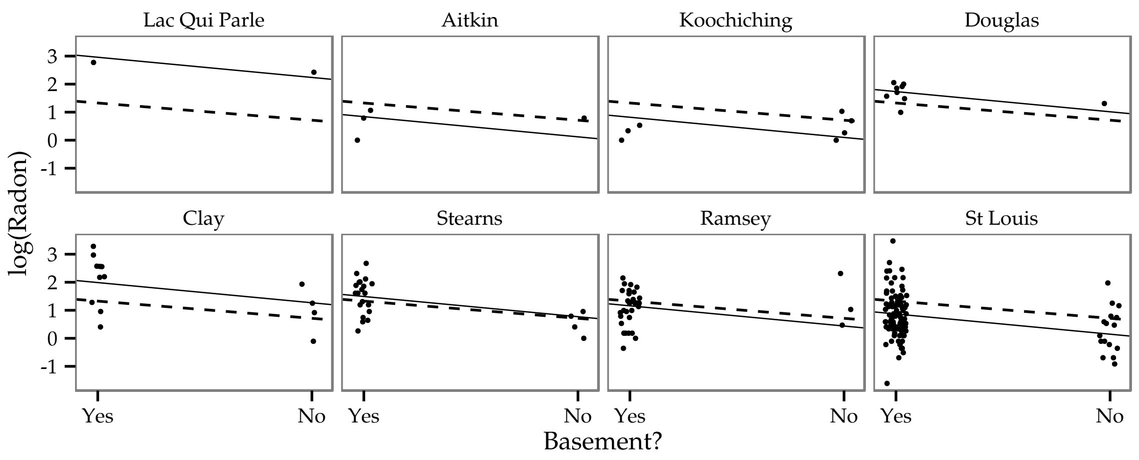
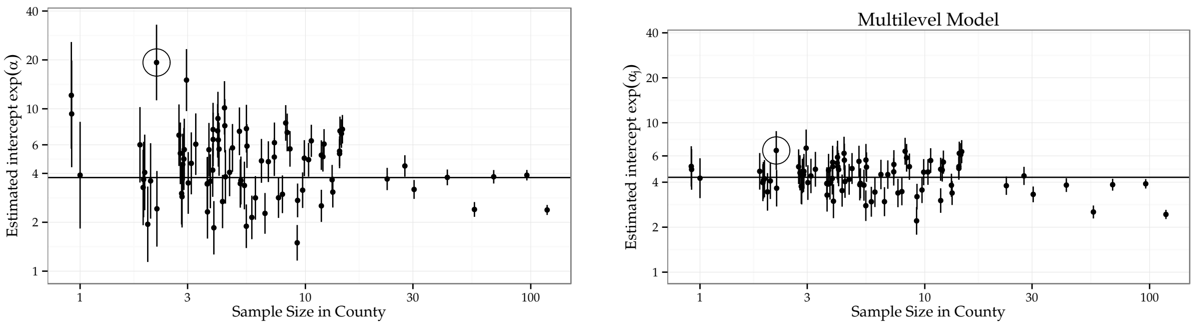


Figure 2: *Complete pooling (dashed lines, $y = \alpha + \beta x$) and no-pooling (solid lines, $y = \alpha_j + \beta x$) regressions fit to 85 counties in Minnesota, and displayed for eight of the counties. The estimated slopes β differ slightly for the two models, but here our focus is on the intercepts.*

```
radon8.data$county.name <{ radon8.data$county
radon8.data$county.name <{ factor(radon8.data$county.name,
                                  levels=c("36","1","35","21","14","71","61","70"),
                                  labels=c("Lac Qui Parle", "Aitkin",
                                           "Koochiching", "Douglas", "Clay",
                                           "Stearns", "Ramsey", "St Louis"))
radon8.data$pooled.int <{ coef(lm.pooled)[1]
radon8.data$pooled.slope <{ coef(lm.pooled)[2]
radon8.data$unpooled.int <{ coef(lm.unpooled)[-1][radon8.data$county]
radon8.data$unpooled.slope <{ coef(lm.unpooled)[1]

p2 <{ ggplot(radon8.data, aes(x, y)) +
  geom_jitter(position = position_jitter(width = .05, height = 0), size=I(1)) +
  scale_x_continuous(breaks=c(0,1), labels=c("Yes", "No")) +
  geom_abline(aes(intercept = pooled.int, slope = pooled.slope),
              linetype = "dashed") +
  geom_abline(aes(intercept = unpooled.int, slope = unpooled.slope),
              size = 0.25) +
  facet_wrap(  county.name, ncol = 4) +
  theme_bw() +
  theme(panel.grid=element_blank(), strip.background = element_blank(),
        axis.title = element_text(size=10), axis.text = element_text(size=8),
        strip.text = element_text(size=8)) +
  labs(x="Basement?", y = "log(Radon)")
print(p2)
```

The code below produces a plot of the radon concentrations corresponding to the intercepts α and α_j for the pooled and unpooled data, respectively:

```
# Fig. 3(a)
unpooled <{ coef(lm.unpooled)[-1]

sd.unpooled <{ coef(summary(lm.unpooled))[-1,'Std. Error']
frame3a <{ data.frame(x = sample.size.jittered, y = exp(unpooled),
                     y.min = exp(unpooled - sd.unpooled),
                     y.max = exp(unpooled + sd.unpooled) )
frame3a.lqp <{ data.frame(x = frame3a$x[lqp.index],
                         y = frame3a$y[lqp.index] )
p3a <{ ggplot(frame3a,aes(x = x, y = y)) +
  geom_point( aes(x = x, y = y), data = frame3a.lqp, shape = I(1),
              size = I(10) ) +
  geom_pointrange(aes(ymin = y.min, ymax = y.max, y=y)) +
```



(a) *Unpooled* **(b)** *Partially pooled*

Figure 3: *Analysis of the intercepts (i.e., radon concentrations for houses with basements) for no-pooling versus multilevel (partial pooling) models. The horizontal lines correspond to the state average in each model. Lac Qui Parle county is circled. As with Fig. 1, we see that no-pooling models tend to exaggerate the variation between counties, thus making it appear that come counties with few measurements have exceptionally high and low radon concentrations.*

```
  geom_hline(yintercept=exp(coef(lm.pooled)['(Intercept)'])) +
  scale_y_log10(expression(paste("Estimated intercept ", exp(alpha))),
                limits = y.limits,
                breaks=c(1,2,4,6,10, 20, 40)) +
  scale_x_log10("Sample Size in County", breaks=c(1,3,10,30,100,300)) +
  #  labs(title = 'No Pooling') +
  theme_bw()
print(p3a)
```

The same problems that we saw with Fig. 1a show up in Fig. 2: The completely pooled analysis ignores differences between counties, and the whole point of this project is to identify counties that have particularly high levels of radon. Conversely, the completely un-pooled analysis overstates the variation from one county to another, especially for counties with only a few measurements, such as Lac Qui Parle.

2.3.1 JAGS model for modeling an individual-level predictor

We can remedy this problem with the following JAGS model, which is very similar to the previous one, but includes a term β for the effect of having a basement:

```
## @knitr radon_multilevel_pred_jags
# FILE: radon.multilevel.pred.jags
#
# JAGS  code for multilevel model for radon
#
model {
  for (i in 1:n){
    y[i]   dnorm (y.hat[i], tau.y)
    y.hat[i] <{ a[county[i]] + b * x[i]
  }
  b   dnorm(0,0.0001) # sigma = 100
  tau.y <{ pow(sigma.y, -2)
  sigma.y   dunif (0, 100)

  for (j in 1:n.counties){
    a[j]   dnorm (mu.a, tau.a)
  }
  mu.a   dnorm (0, .0001) # sigma = 100
  tau.a <{ pow(sigma.a, -2)
  sigma.a   dunif (0, 100)
}
```

2.3.2 R code for modeling an individual-level predictor

The R code below runs the JAGS model and produces Fig. 3.

```
# We need to initialize the parameters for the model.
# Initialize alpha_j, beta, and mu randomly from normal(0,1) distributions
# Initialize sigma randomly from a uniform distribution on [0,1]
predictor.inits <{ function (chain){
  seed <{ jg.seeds[(100 + chain) %% length(jg.seeds) + 1]
  list (a=rnorm(n.counties), b = rnorm(1),
        mu.a=rnorm(1), sigma.y=runif(1), sigma.a=runif(1)
  )
}
# prepare a list of data to pass to JAGS model
radon.data <{ list (n = n, n.counties = n.counties, y = y, county = county,
                    x = x)
# Tell JAGS the names of the parameters it should report back to us
radon.parameters <{ c ("a", "b", "mu.a", "sigma.y", "sigma.a")

# Compile the JAGS model, initializing the data and parameters
mlm.radon.pred.model <{ jags.model("radon_multilevel_pred.jags",
                                   data = radon.data,
                                   inits = predictor.inits,
                                   n.chains = 3,
                                   n.adapt=1000)

# After warming up, take 2000 random samples in each of 3 chains.
update(mlm.radon.pred.model,n.iter=2000)
mlm.radon.pred <{ coda.samples( mlm.radon.pred.model,
                                variable.names = radon.parameters,
                                n.iter = 2000)

# Here, we get the data back from JAGS and convert it to a useful form
post.pred <{ as.matrix(mlm.radon.pred)
alphabarbar <{ mean(post.pred[,'mu.a'])
mean.a.pred <{ rep (NA, n.counties)
sd.a.pred <{ rep (NA, n.counties)
for (i in 1:n.counties) {
  mean.a.pred[i] <{ mean(post.pred[ , paste('a[',i,']', sep='')])
  sd.a.pred[i] <{ sd(post.pred[ , paste('a[',i,']', sep='')])
}
```

The R code below plots the results of the JAGS analysis to produce Fig. 3b.

```
# Fig. 3(b)
frame3b <{ data.frame( x = sample.size.jittered, y = exp(mean.a.pred),
                      y.min = exp(mean.a.pred - sd.a.pred),
                      y.max = exp(mean.a.pred + sd.a.pred))
frame3b.lqp <{ data.frame( x= frame3b$x[lqp.index],
                          y= frame3b$y[lqp.index] )
p3b <{ ggplot( frame3b, aes(x = x, y = y) ) +
  geom_point( aes(x = x, y = y), data = frame3b.lqp, shape = I(1),
              size = I(10) ) +
  scale_y_log10(expression(paste("Estimated intercept ", exp(alpha[j]))),
                limits = y.limits,
                breaks=c(1,2,4,6,10, 20, 40)) +
  scale_x_log10("Sample Size in County", breaks=c(1,3,10,30,100,300)) +
  geom_pointrange(aes(ymin = y.min, ymax = y.max, y = y)) +
  geom_hline(yintercept=exp(alphabarbar)) +
  theme_bw() +
  labs(title="Multilevel Model")

print(p3b)
```

We can also repeat the eight-county plot from Fig. 2 to produce Fig. 4, which shows the regression against the presence of a basement for our multilevel partially pooled model.

```
# Fig. 4
radon8.data$pooled.int <{ coef(lm.pooled)[1]
```

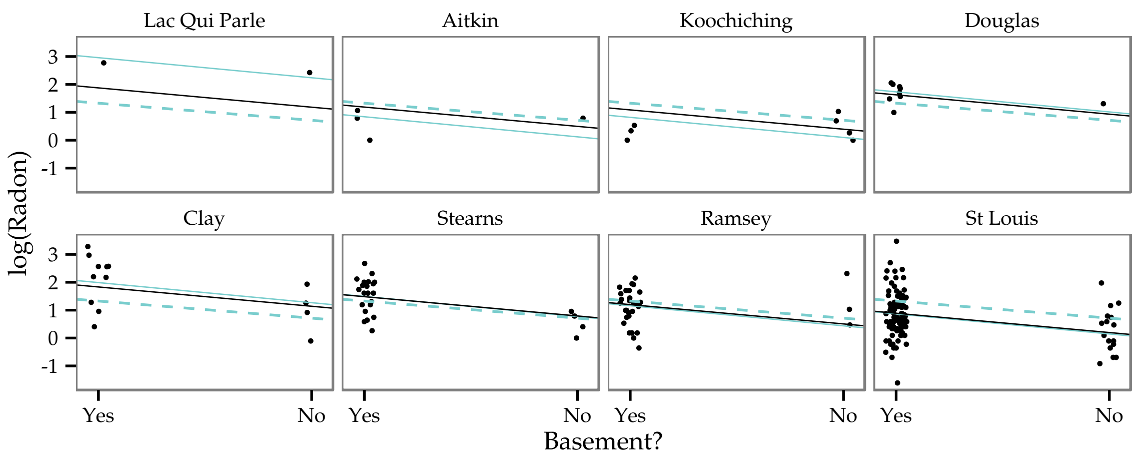


Figure 4: *Multilevel (partial-pooling) regression lines ($y = \alpha_j + \beta x$) fit to radon data from Minnesota, and displayed for eight counties. Light blue solid and dashed lines show the complete-pooling and no-pooling estimates, respectively, from Fig. 2.*

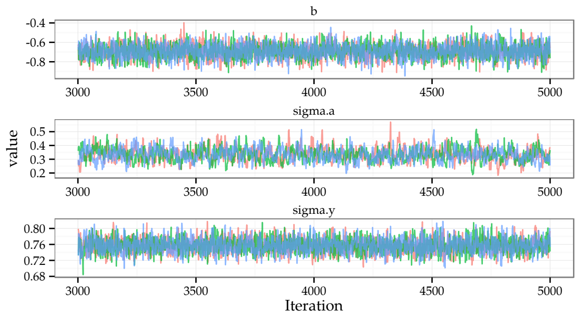
```
radon8.data$pooled.slope <{ coef(lm.pooled)[2]
radon8.data$unpooled.int <{ coef(lm.unpooled)[-1][radon8.data$county]
radon8.data$unpooled.slope <{ coef(lm.unpooled)[1]

radon8.data$mlm.int <{ mean.a.pred[radon8.data$county]
radon8.data$mlm.slope <{ mean(post.pred[,'b'])

p4 <{ ggplot(radon8.data, aes(x, y)) +
  geom_jitter(position = position_jitter(width = .05, height = 0), size=I(1)) +
  scale_x_continuous(breaks=c(0,1), labels=c("Yes", "No")) +
  geom_abline(aes(intercept = pooled.int, slope = pooled.slope),
              linetype = "dashed", col=I('darkslategray3')) +
  geom_abline(aes(intercept = unpooled.int, slope = unpooled.slope),
              size = 0.25, col=I('darkslategray3')) +
  geom_abline(aes(intercept = mlm.int, slope = mlm.slope),
              size=0.25, col=I('black')) +
  facet_wrap(  county.name, ncol = 4) +
  facet_wrap(  county.name, ncol = 4) +
  theme_bw() +
  theme(panel.grid=element_blank(), strip.background = element_blank(),
        axis.title = element_text(size=10), axis.text = element_text(size=8),
        strip.text = element_text(size=8)) +
  labs(x="Basement?", y = "log(Radon)")
print(p4)
```

3 Diagnosing JAGS analysis

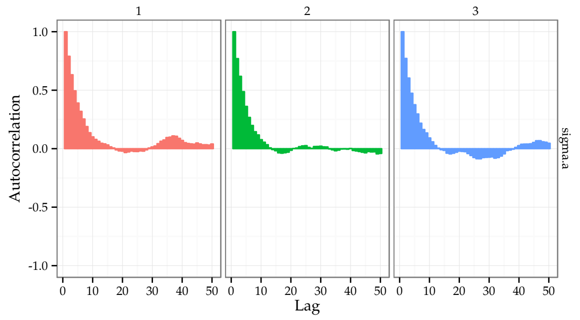
This section closely follows Chapter 16 of Gelman & Hill 2006.

JAGS (and its technique of Gibbs-Sampling Markov-Chain Monte Carlo integration) may seem like magic, especially since we're not digging into the guts of how it works, but while it is a powerful technique, it does not work perfectly all the time, and it is important to understand how to diagnose the output and check whether it makes sense.

Monte Carlo analysis is the name of a big family of techniques for optimization and numerical integration. It was invented by Stanislaw Ulam, Enrico Fermi, Nicholas Metropolis, and Stanley Frankel in the mid-1940s as part of the U.S. nuclear weapons research program. It was named by John von Neumann, who said that the use of random numbers reminded him of Ulam's uncle gambling at the Monte Carlo casino in Monaco (Metropolis 1987) The essence of the method is that



(a) *Mixing for σ_a, σ_y, and β.*



(b) *Autocorrelation for σ_a.*

Figure 5: *Mixing and autocorrelation analysis Monte Carlo calculation of the multilevel model for partially-pooled counties with a predictor for the basement. (**a**) shows mixing for σ_a, σ_y, and β, calculated with three chains, and (**b**) shows autocorrelation analysis for 3 chains of σ_a.*

when a mathematical computation, such as calculating a probability or an integral, is too hard to solve exactly, a good approximation can be achieved using random numbers. Basically, you draw a random number and use it as the basis of a calculation of one piece of the equation, and then draw another random number and repeat the calculation. In what follows, we will refer to drawing a random number and performing the corresponding calculation as an **iteration**.

After a large number of iterations, it is possible to combine the calculated parts to form an approximate solution to the original equation. The more iterations you perform, the more accurate your approximation will be, and computers make it efficient to perform many iterations. But whether you are calculating by hand or with a computer, you need to know when to stop.

There is no automatic way to tell whether you have performed enough iterations. It is always possible that the Monte Carlo calculation is wildly wrong. We can't prove that a calculation is accurate, but there are signs of a calculation going badly. If these signs are not present, it's still possible that the calculation is inaccurate, but if the signs are present, then it is almost certain that the calculation is inaccurate.

Three diagnostics that I will describe here are mixing, autocorrelation, and the Gelman-Rubin potential scale reduction factor, called $\hat{R}$ (R-hat).

3.1 Mixing

In Monte Carlo analysis, we start with a randomly chosen value for each parameter in the equation for the posterior probability and at each iteration, we randomly change the parameters. If there are N parameters, their variations define an N-dimensional space (called "phase space" or "parameter space"), and for the Monte Carlo calculation to produce a reasonably accurate answer, the random variation of the parameters has to cover a good portion of the relevant part of that phase-space. One way to tell whether the calculation has done so is to start several different sequences of iterations from different starting points in phase space (i.e., different sets of values for the parameters) and see whether, after a warm-up period, they all seem to cover the same parts of phase space. We can do this with the following code, which produces Fig. 5a:

```
# Fig. 5(a)
samples <{ ggs(mlm.radon.pred, family='(sigma|b).*')
ggs_traceplot(samples) +
  theme_bw() + theme(legend.position='none', strip.background = element_blank())
```

We see the results in Fig. 5a, which shows how the three chains (red, blue, and green) vary over 2000 iterations (the first 1000 were used to warm up the Monte Carlo routine, and are not used for calculating the parameters). We are looking for good overlap of the chains, which we see here. If the chains do not overlap well, and if some chains seem to be spending a disproportionate amount of

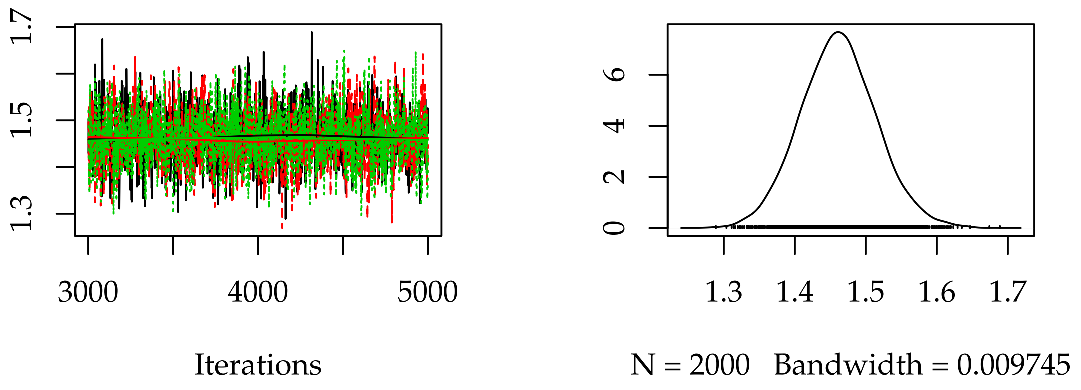


Figure 6: *Mixing analysis and posterior probability distribution for μ_a. At the bottom of the posterior probability plot, you can see little tick-marks representing the value of μa at each individual iteration in the Monte Carlo calculation*

time at certain values, rather than covering the whole range that the other chains do, it would be a cause for concern. See Fig. 11.3 on p. 283 of Gelman *et al.* 2013 for examples of poorly mixed chains.

Fig. 6 shows another, similar, diagnostic plot, which juxtaposes a traceplot of the chains for μ_a (to check mixing) and a posterior probability plot for μ_a.

```
# Fig. 6
plot(mlm.radon.pred[,'mu.a'])
```

3.2 Autocorrelation and Thinning

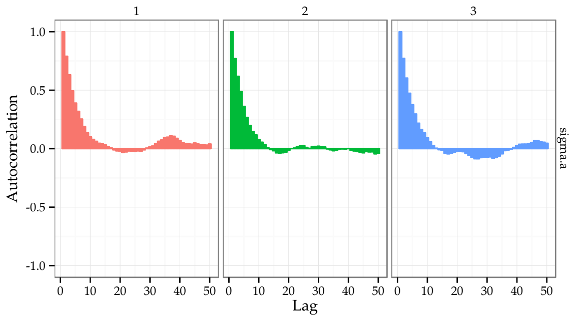
In Fig 5b, we look at the autocorrelation between subsequent iterations.

```
# Fig. 5(b)
auto.plot <{ ggs_autocorrelation(samples, family='sigma.a') +
  theme_bw() + theme(legend.position='none', strip.background = element_blank())
auto.plot
```

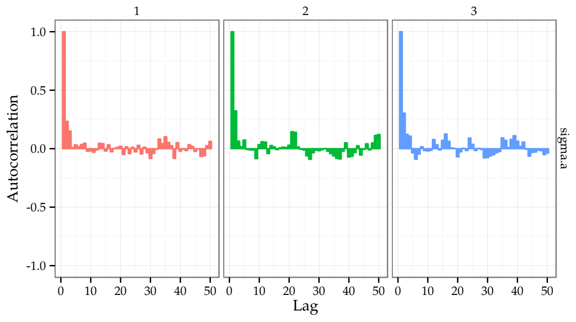
A well-mixed Monte Carlo chain should show no correlation between subsequent iterations. Here, we see that there is very little autocorrelation for lags greater than 3 or so, but we might want to be concerned about the lag-1–3 autocorrelation. This doesn't present a problem by itself, but simply means that the effective number of independent iterations is around one quarter the actual number. We can either sample around four times as many iterations to compensate for the autocorrelation or we can "thin" the Monte Carlo by telling it to remember only every fourth iteration:

```
# Fig. 7
thin.steps = 4
  mlm.radon.pred2 <{ coda.samples( mlm.radon.pred.model,
                                   variable.names = radon.parameters,
                                   n.iter = 2000, thin=thin.steps)
auto.plot.thinned <{ ggs_autocorrelation(ggs(mlm.radon.pred2),
                                         family='sigma.a') +
  theme_bw() + theme(strip.background = element_blank(), legend.position='none')
print(auto.plot)
print(auto.plot.thinned)
```

Fig. 7 compares the autocorrelation for the un-thinned Monte Carlo calculation, and for thinning by 4 (thinning by n means we keep one of every n iterations).



(a) *No thinning*



(b) *Thinning factor = 4*

Figure 7: *Autocorrelation for MCMC with no thinning and thinning factor of 4 (3 chains each)*

3.3 Gelman-Rubin potential scale-reduction factor (PSRF)

Andrew Gelman and Donald Rubin devised a diagnostic measure that predicts how much more accurate the estimate of a parameter's value would be if you ran an infinite number of iterations. This is called the **potential scale-reduction factor**, or **PSRF** (Gelman *et al*. 2013, p. 285). If PSRF = 1, then the uncertainty in your estimate of the parameter would not get any smaller no matter how many more iterations you run. For PSRF > 1, the uncertainty could potentially reduce by a factor of PSRF − 1: for example, if PSRF = 1.3, then running more iterations could potentially make your estimate of the parameter 30% more accurate.

```
gelman.diag(mlm.radon.pred[,c('mu.a','sigma.a','sigma.y','b')])
```

```
Potential scale reduction factors:

        Point est. Upper C.I.
mu.a             1       1.01
sigma.a          1       1.00
sigma.y          1       1.00
b                1       1.00

Multivariate psrf

1
```

Here, we see that for μ_a, σ_a, σ_y, and β, the best we could do by running lots more iterations would be to increase our accuracy of σ_a and σ_y by around 1%.

4 Group-level predictors

We have looked at the impact of accounting for house-level variation in the data, due to the presence of a basement, and now we will look at explaining county-level variations due to the amount of uranium in the soil:

$$y_i \sim \text{Normal}(a_{j[i]} + \beta x_i, \sigma_y^2) \tag{5}$$

$$a_j \sim \text{Normal}(\gamma_0 + \gamma_1 u_j, \sigma_a^2), \tag{6}$$

where u_j is the logarithm of the county-average concentration of uranium in the soil in county j.

4.0.1 JAGS code for group-level predictors

The JAGS code below models these equations, with g0 and g1 representing γ_0 and γ_1:

```
## @knitr radon_multilevel_grp_jags
# FILE: radon.multilevel.grp.jags
#
# JAGS  code for multilevel model for radon
#
model {
  for (i in 1:n){
    y[i]   dnorm (y.hat[i], tau.y)
    y.hat[i] <{ a[county[i]] + b * x[i]
  }
  b   dnorm(0,0.0001) # sigma = 100
  tau.y <{ pow(sigma.y, -2)
  sigma.y   dunif (0, 100)

  for (j in 1:n.counties){
    a[j]   dnorm (g0 + g1 * u[j], tau.a)
  }
  g1   dnorm(0,0.0001) # sigma = 100
  g0   dnorm (0, .0001) # sigma = 100
  tau.a <{ pow(sigma.a, -2)
  sigma.a   dunif (0, 100)
}
```

4.0.2 R code for modeling group-level predictors

This R code loads the uranium data:

```
## Get the county-level predictor
srrs2.fips <{ srrs2$stfips*1000 + srrs2$cntyfips
cty <{ read.table ("cty.dat", header=T, sep=",")
usa.fips <{ 1000*cty[,"stfips"] + cty[,"ctfips"]
usa.rows <{ match (unique(srrs2.fips[mn]), usa.fips)
uranium <{ cty[usa.rows,"Uppm"]
u <{ log (uranium)

u.full <{ u[county]
```

And this R code runs the model:

```
# We need to initialize the parameters for the model.
# Initialize alpha_j, beta, gamma_0, and gamma_1 randomly
# from normal(0,1) distributions.
# Initialize sigma randomly from a uniform distribution on [0,1]
group.inits <{ function (chain){
  list (a=rnorm(n.counties), b = rnorm(1),
        g0 = rnorm(1), g1=rnorm(1),
        sigma.y=runif(1), sigma.a=runif(1)
  )
}
# prepare a list of data to pass to JAGS model
group.data <{ list (n = n, n.counties = n.counties, y = y, county = county,
                  x = x, u = u)
# Tell JAGS the names of the parameters it should report back to us
group.parameters <{ c ("a", "b", "g0", "g1", "sigma.y", "sigma.a")

# Compile the JAGS model, initializing the data and parameters
mlm.radon.grp.model <{ jags.model("radon_multilevel_grp.jags",
                                data = group.data,
                                inits = group.inits,
                                n.chains = 3,
                                n.adapt=1000)

# After warming up, take 2000 random samples in each of 3 chains.
update(mlm.radon.grp.model,n.iter=2000)
mlm.radon.group <{ coda.samples( mlm.radon.grp.model,
                               variable.names = group.parameters,
                               n.iter = 2000)
```

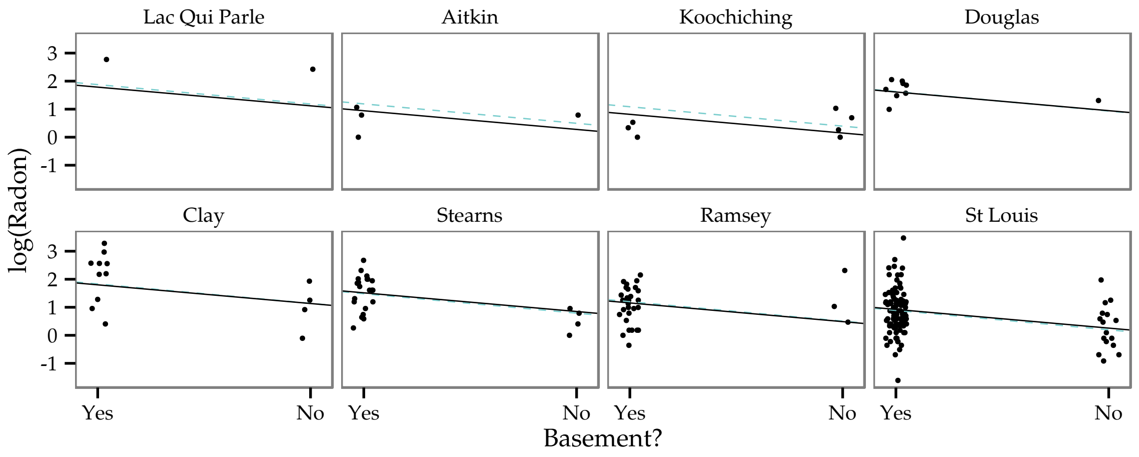


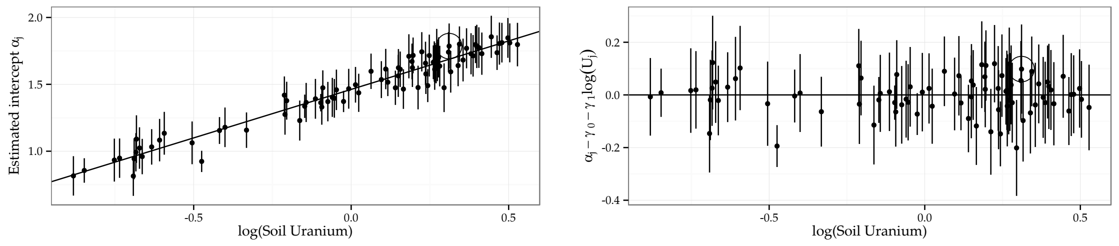
Figure 8: *Multilevel (partial-pooling) regression lines ($y = \alpha_j + \beta x$) fit to radon data from Minnesota, and displayed for eight counties, including soil concentration of uranium as a county-level predictor. Light blue dashed lines show the multilevel estimates, without uranium as a predictor, from Fig. 4.*

```
# Here, we get the data back from JAGS and convert it to a useful form
post.grp <{ as.matrix(mlm.radon.group)
avg.int <{ mean(post.grp[,'g0'])
avg.slope <{ mean(post.grp[,'g1'])
sigma.y.grp <{ mean(post.grp[,'sigma.y'])
sigma.a.grp <{ mean(post.grp[,'sigma.a'])
b.grp <{ mean(post.grp[,'b'])
mean.a.grp <{ rep (NA, n.counties)
sd.a.grp <{ rep (NA, n.counties)
for (i in 1:n.counties) {
mean.a.grp[i] <{ mean(post.grp[ , paste('a[',i,']', sep='')])
sd.a.grp[i] <{ sd(post.grp[ , paste('a[',i,']', sep='')])
}
```

4.0.3 R code to plot results from group-level model

This code produces Fig. 8, showing the regressions against basements with uranium as a county-level predictor.

```
# Fig. 8
radon8.data$grp.int <{ mean.a.grp[radon8.data$county]
radon8.data$grp.slope <{ mean(post.grp[,'b'])

p8 <{ ggplot(radon8.data, aes(x, y)) +
  geom_jitter(position = position_jitter(width = .05, height = 0), size=I(1)) +
  scale_x_continuous(breaks=c(0,1), labels=c("Yes", "No")) +
  geom_abline(aes(intercept = mlm.int, slope = mlm.slope),
              size = 0.25, col=I('darkslategray3'), linetype='dashed') +
  geom_abline(aes(intercept = grp.int, slope = grp.slope),
              size=0.25, col=I('black')) +
  facet_wrap(  county.name, ncol = 4) +
  facet_wrap(  county.name, ncol = 4) +
  theme_bw() +
  theme(panel.grid=element_blank(), strip.background = element_blank(),
        axis.title = element_text(size=10), axis.text = element_text(size=8),
        strip.text = element_text(size=8)) +
  labs(x="Basement?", y = "log(Radon)")
print(p8)
```



(a) *Intercepts: α_j.* **(b)** *Residuals.*

Figure 9: *Model with group-level and individual-level predictors: County-level soil-uranium concentration and household-level basement. Lac Qui Parle county is circled.*

And here, we plot the fitted intercepts against the county-level uranium (Fig. 9):

```
# Fig. 9
frame9 <{ data.frame( x = u, y = mean.a.grp,
                      y.min = mean.a.grp - sd.a.grp,
                      y.max = mean.a.grp + sd.a.grp)
frame9.lqp <{ frame9[lqp.index,]
p9a <{ ggplot( frame9, aes(x = x, y = y) ) +
  geom_point( aes(x = x, y = y), data = frame9.lqp, shape = I(1),
              size = I(10) ) +
  scale_y_continuous(expression(paste("Estimated intercept ", alpha[j]))) +
  scale_x_continuous("log(Soil Uranium)") +
  geom_pointrange(aes(ymin = y.min, ymax = y.max, y = y)) +
  geom_abline(intercept=avg.int, slope=avg.slope)  +
  # labs(title="Multilevel Model with Basement and County-Level Uranium") +
  theme_bw()

print(p9a)

p9b <{ ggplot(frame9, aes(x=x, y=y - avg.slope * x)) +
  geom_pointrange(aes(x=x,y=y-avg.slope*x-avg.int,
                      ymin=y.min-avg.slope*x-avg.int,
                      ymax=y.max-avg.slope*x-avg.int)) +
  geom_hline(yintercept=0) +
  geom_point(aes(x=x,y=y-x*avg.slope-avg.int),
             data=frame9.lqp, shape=I(1), size=I(10)) +
  scale_y_continuous(expression(alpha[j] - gamma[0] - gamma[1] * log(U[j]))) +
  scale_x_continuous("log(Soil Uranium)") +
  theme_bw()
# +  labs(title='Multi')

print(p9b)
```

5 Testing models with fake data

In Section 3, we learned about ways to test whether the JAGS Monte-Carlo calculation converged, but even if the calculation converged, the model might not accurately find the correct parameters. Here, we try another approach to testing the models: We generate "fake data" by simulating the process we're modeling using known parameters, and then we analyze the fake data using our model and see whether we accurately recover the parameters we used to generate the data.

5.1 Generating fake data

We pick some arbitrary values for the parameters. The values we fit to our real data are: $\gamma_0 = 1.46$, $\gamma_1 = 0.73$, $\sigma_y = 0.76$, $\sigma_a = 0.15$, and $\beta = -0.67$.

```
fake.g0 <{ 1.6
fake.g1 <{ 0.5
fake.b <{ -0.8
fake.sigma.y <{ 0.9
fake.sigma.a <{ 0.2
```

Now we generate some arbitrary predictors for 150 measurements spread across 15 fictitious counties:

```
fake.n.counties <{ 15
fake.n <{ 10 * fake.n.counties
  fake.county <{ sample(fake.n.counties,fake.n,T)
fake.u <{ log(rnorm(fake.n.counties, mean(uranium), sd(uranium)))
fake.x <{ rbinom(fake.n,1,mean(x)) # 0 for basement, 1 for ground floor
```

Now generate some fake data:

$$a_j \sim \text{Normal}(\gamma_0 + \gamma_1 u_j, \sigma_a) \tag{7}$$
$$y_i \sim \text{Normal}(\alpha_{\text{county}_i} + \beta x, \sigma_y) \tag{8}$$

```
fake.alpha <{ fake.g0 + fake.g1 * fake.u +
  rnorm(fake.n.counties, 0, fake.sigma.a)
fake.y <{ rep(NA,fake.n)
for (i in 1:fake.n) {
  fake.y[i] <{ fake.alpha[fake.county[i]] + fake.b * fake.x[i] +
    rnorm(1, 0,  fake.sigma.y)
}
```

5.2 Fitting a model to fake data

Now we fit the model to the data. We use the same model as in Section 4, with both individual (household basement) and group-level (county soil-uranium concentration) predictors.

```
# We need to initialize the parameters for the model.
# Initialize alpha_j, beta, gamma_0, and gamma_1 randomly
# from normal(0,1) distributions.
# Initialize sigma randomly from a uniform distribution on [0,1]
fake.inits <{ function(chain) {
  list(a=rnorm(fake.n.counties), b = rnorm(1),
       g0 = rnorm(1), g1 = rnorm(1),
       sigma.y = runif(1), sigma.a = runif(1)
  )
}
fake.data <{ list(n = fake.n, n.counties = fake.n.counties, county = fake.county,
                x = fake.x, u = fake.u, y = fake.y)
fake.model <{ jags.model("radon_multilevel_grp.jags",
                       data = fake.data,
                       inits = fake.inits,
                       n.chains = 4,
                       n.adapt=1000)

#
# Sigma_a has a hard time converging,
# so use a long burn-in and a long sample run.
#
update(fake.model, niter=4000)
# After warming up, take 2000 random samples in each of 3 chains.
fake.samples <{ coda.samples( fake.model,
                            variable.names = group.parameters,
                            n.iter = 4000)
```

After running JAGS, we should always check that the chains have converged, so check $\hat{R}$:

```
gelman.diag(fake.samples)
```

```
Potential scale reduction factors:

        Point est. Upper C.I.
a[1]          1.00       1.00
a[2]          1.00       1.00
a[3]          1.00       1.00
a[4]          1.00       1.01
a[5]          1.00       1.00
a[6]          1.00       1.01
a[7]          1.00       1.00
a[8]          1.00       1.00
a[9]          1.00       1.01
a[10]         1.00       1.01
a[11]         1.00       1.01
a[12]         1.01       1.02
a[13]         1.00       1.01
a[14]         1.00       1.00
a[15]         1.00       1.00
b             1.00       1.00
g0            1.00       1.01
g1            1.00       1.00
sigma.a       1.02       1.05
sigma.y       1.00       1.00

Multivariate psrf

1.01
```

5.3 Interpreting model output from fake-data

Now we can look at the posterior distributions for the parameters in our model to see whether it does a good job of recovering the parameters we used to generate our fake data. If the model can recover the parameters accurately from fake data, where we know what the parameters are, then we will have more confidence in parameters estimated from real data. Figs. 10 and 11 and Table 1 show various aspects of our posterior probabilty distributions for the 20 parameters and how they compare to the true values. Note that for the confidence intervals, I use the 50% highest-density intervals (HDI) on the posterior instead of some scaling factor times the standard deviation. Highest-density intervals are the smallest interval around the median that contains a given fraction of the total probability (Kruschke 2012, 2013a,b) For a unimodal distribution, this is uniquely defined, but for a multimodal distribution, it may not be.

I use HDI insted of standard deviations because the histograms of the posterior are non-normal (and very skewed for several of the parameters, such as σ_a, shown in Fig. 11), so mean ± standard deviation may not be a good measure of a true confidence interval. Note how, in Fig. 11, the mean of the posterior is not at the center of the HDI. You can also see examples of similarly skewed posterior distributions in Fig. 10, where the heavy black lines for the 50% HDI are very asymmetric relative to the dot representing the mean of the posterior for σ_a, a_4, a_6, a_9, a_{10}, a_{12}, and a_{13}.

Table 1 and Figure 10 show that 7 of the 20 parameters (35%) are within the 50% confidence interval, and 20 of the 20 parameters (100%) within the 95% HDI. This reassures us that our fit is working appropriately on the fake data (Note that the probability of exactly 10 of 20 parameters lying within the 50% HDI is 18%, so we don't expect exactly 10 to lie within the HDI every time, but we do expect, with 88% probability, that between 7 and 13 will lie in the HDI if our model-fitting procedure is working properly).

If we really want to check the accuracy of our approch, we would repeat this fake-data exercise many times, with different random values for the fake data and check that on average, 50% of the 50% HDI's overlap the correct values of the parameters. When I was writing this up, I had a problem where only about 35% of the HDI's contained the correct values of the parameters, so I suspected a bug in my R code and this helped me track it down (it was a typo that caused me to pass the wrong initialization function to the JAGS model for the fake data).

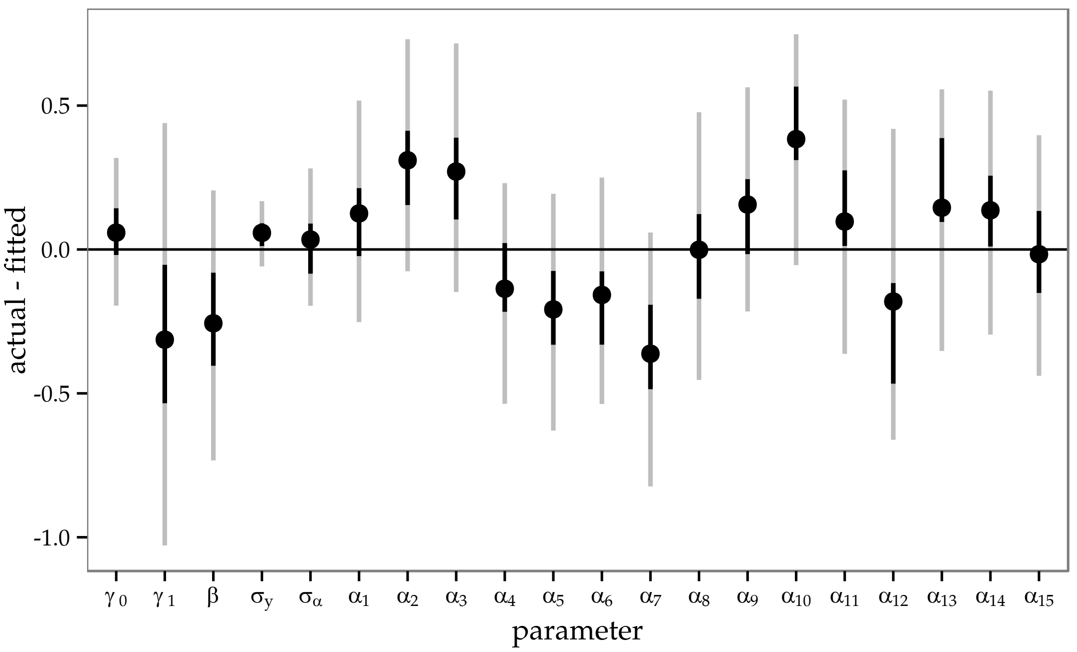


Figure 10: *Fake data fit errors. The heavy black lines show the 50% highest-density interval (HDI) of the posterior distribution, the light gray lines show the 95% HDI, and the dots show the mean of the posterior. We expect that 50% of the black lines and 95% of the gray lines should overlap zero.*

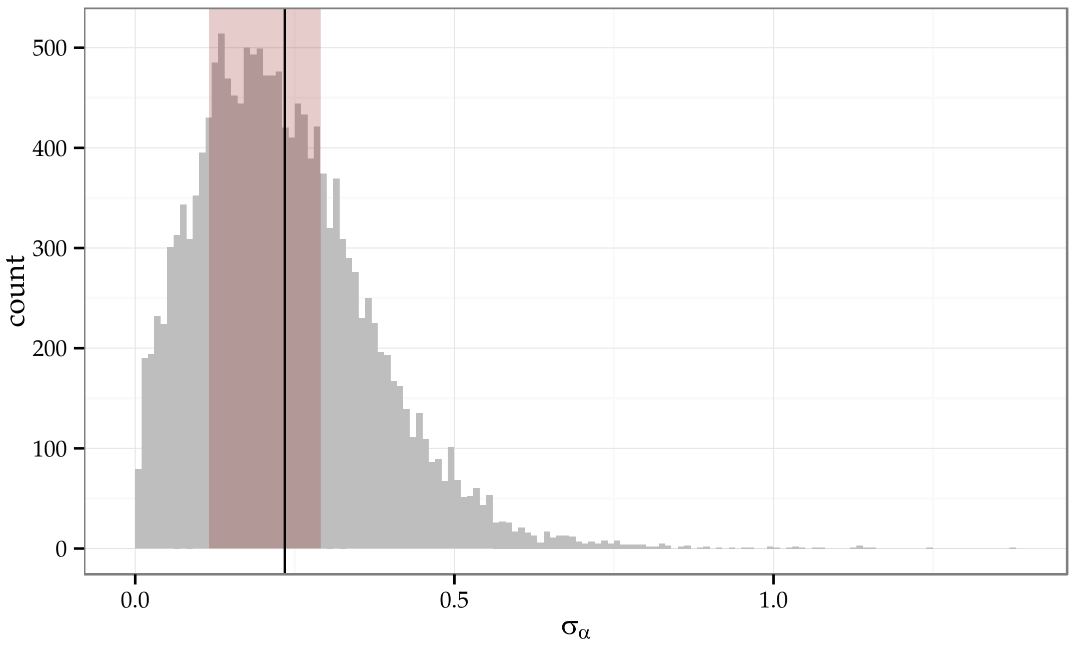


Figure 11: *Histogram of the posterior distribution of σ_α. The vertical line shows the mean of the distribution. The shaded area shows the 50% HDI. Note that the mean is not at the center of the HDI.*

		Highest Density Interval (HDI)							
	actual	**lower**		**mean**	**upper**		**actual − fitted**	**scaled error**	
		95%	**50%**		**50%**	**95%**		**50%**	**95%**
γ_0	1.60	1.40	1.58	1.66	1.74	1.92	0.06	0.75	0.23
γ_1	0.50	−0.53	−0.03	0.19	0.45	0.94	−0.31	**1.20**	0.42
β	−0.80	−1.53	−1.20	−1.06	−0.88	−0.59	−0.26	**1.46**	0.56
σ_y	0.90	0.84	0.91	0.96	0.99	1.07	0.06	**1.26**	0.49
σ_a	0.20	0.00	0.12	0.23	0.29	0.48	0.03	0.29	0.15
a_1	1.58	1.33	1.56	1.71	1.80	2.10	0.13	0.85	0.33
a_2	1.40	1.33	1.56	1.71	1.81	2.13	0.31	**1.99**	0.80
a_3	1.39	1.24	1.49	1.66	1.77	2.10	0.27	**1.63**	0.65
a_4	1.71	1.17	1.49	1.57	1.73	1.94	−0.14	0.86	0.37
a_5	1.82	1.19	1.49	1.61	1.74	2.01	−0.21	**1.56**	0.52
a_6	2.01	1.47	1.68	1.85	1.93	2.26	−0.16	**1.93**	0.39
a_7	2.00	1.17	1.51	1.63	1.80	2.05	−0.36	**2.13**	0.86
a_8	1.87	1.41	1.70	1.87	1.99	2.35	0.00	0.01	0.00
a_9	1.62	1.41	1.61	1.78	1.87	2.19	0.16	0.91	0.42
a_{10}	1.16	1.11	1.47	1.54	1.73	1.91	0.38	**5.26**	0.88
a_{11}	1.44	1.08	1.46	1.54	1.72	1.97	0.10	**1.14**	0.21
a_{12}	2.18	1.52	1.71	2.00	2.06	2.60	−0.18	**2.82**	0.30
a_{13}	1.38	1.02	1.47	1.52	1.76	1.93	0.15	**2.93**	0.29
a_{14}	1.56	1.27	1.57	1.70	1.82	2.11	0.14	**1.08**	0.32
a_{15}	1.76	1.32	1.61	1.74	1.89	2.16	−0.02	0.11	0.04

Table 1: *Fake data parameters and fitted values. The column "scaled error" represents the ratio of the error (actual − fitted) to the HDI. If the scaled error is greater than 1, then the actual value lies outside the HDI, and is marked in boldface.*

5.3.1 R code for analyzing the fake data

The code below processed the output of the JAGS run on the fake data. It extracts the parameters, analyzes their distributions to find the 50% and 95% HDI, and generally turns them into useful forms:

```
# Here, we get the data back from JAGS and convert it to a useful form
post.fake <{ as.matrix(fake.samples)
fake.mean.g0      <{ mean(post.fake[,'g0'])
fake.hdi.50.g0       <{ hdi(post.fake[,'g0'], credMass=0.50)
fake.hdi.95.g0       <{ hdi(post.fake[,'g0'], credMass=0.95)
fake.mean.g1      <{ mean(post.fake[,'g1'])
fake.hdi.50.g1       <{ hdi(post.fake[,'g1'], credMass=0.50)
fake.hdi.95.g1       <{ hdi(post.fake[,'g1'], credMass=0.95)
fake.mean.sigma.y <{ mean(post.fake[,'sigma.y'])
fake.hdi.50.sigma.y  <{ hdi(post.fake[,'sigma.y'], credMass=0.50)
fake.hdi.95.sigma.y  <{ hdi(post.fake[,'sigma.y'], credMass=0.95)
fake.mean.sigma.a <{ mean(post.fake[,'sigma.a'])
fake.hdi.50.sigma.a  <{ hdi(post.fake[,'sigma.a'], credMass=0.50)
fake.hdi.95.sigma.a  <{ hdi(post.fake[,'sigma.a'], credMass=0.95)
fake.mean.b       <{ mean(post.fake[,'b'])
fake.hdi.50.b        <{ hdi(post.fake[,'b'], credMass=0.50)
fake.hdi.95.b        <{ hdi(post.fake[,'b'], credMass=0.95)

fake.mean.a <{ rep (NA, fake.n.counties)
fake.hdi.50.a    <{ cbind(lower=rep(NA,fake.n.counties),
                          upper=rep(NA,fake.n.counties))
fake.hdi.95.a    <{ cbind(lower=rep(NA,fake.n.counties),
                          upper=rep(NA,fake.n.counties))
for (i in 1:fake.n.counties) {
  fake.mean.a[i] <{ mean(post.fake[ , paste('a[',i,']', sep='')])
  fake.hdi.50.a[i,]    <{ hdi(post.fake[ , paste('a[',i,']', sep='')],
                             credMass=0.50)
  fake.hdi.95.a[i,]   <{ hdi(post.fake[ , paste('a[',i,']', sep='')],
                             credMass=0.95)
}
```

The code below produces Figure 10:

```
# Fig. 10
params <{ data.frame( actual = c(g0 = fake.g0, g1 = fake.g1, b = fake.b,
                                 sigma.y = fake.sigma.y,
                                 sigma.a = fake.sigma.a ),
                      fitted = c(g0 = fake.mean.g0, g1 = fake.mean.g1,
                                 b = fake.mean.b, sigma.y = fake.mean.sigma.y,
                                 sigma.a = fake.mean.sigma.a),
                      lower.50 = c(g0 = fake.hdi.50.g0[1],
                                   g1 = fake.hdi.50.g1[1],
                                   b = fake.hdi.50.b[1],
                                   sigma.y = fake.hdi.50.sigma.y[1],
                                   sigma.a = fake.hdi.50.sigma.a[1]),
                      upper.50 = c(g0 = fake.hdi.50.g0[2],
                                   g1 = fake.hdi.50.g1[2],
                                   b = fake.hdi.50.b[2],
                                   sigma.y = fake.hdi.50.sigma.y[2],
                                   sigma.a = fake.hdi.50.sigma.a[2]),
                      lower.95 = c(g0 = fake.hdi.95.g0[1],
                                   g1 = fake.hdi.95.g1[1],
                                   b = fake.hdi.95.b[1],
                                   sigma.y = fake.hdi.95.sigma.y[1],
                                   sigma.a = fake.hdi.95.sigma.a[1]),
                      upper.95 = c(g0 = fake.hdi.95.g0[2],
                                   g1 = fake.hdi.95.g1[2],
                                   b = fake.hdi.95.b[2],
                                   sigma.y = fake.hdi.95.sigma.y[2],
                                   sigma.a = fake.hdi.95.sigma.a[2]) )

params <{ rbind(params, data.frame(actual = fake.alpha,
                                   fitted = fake.mean.a,
                                   lower.50 = fake.hdi.50.a[,1],
                                   upper.50 = fake.hdi.50.a[,2],
                                   lower.95 = fake.hdi.95.a[,1],
                                   upper.95 = fake.hdi.95.a[,2],
                                   row.names = paste('a[', 1:fake.n.counties,
                                                     ']',sep='') ))
params$delta <{ params$fitted - params$actual
params$std.res.50 <{ with(params, ifelse(delta > 0,
                                         abs(delta / (fitted-lower.50)),
                                         abs(delta/(fitted-upper.50))))
params$std.res.95 <{ with(params, ifelse(delta > 0,
                                         abs(delta / (fitted-lower.95)),
                                         abs(delta/(fitted-upper.95))))
xlabs <{ unlist(list(expression(gamma[0]), expression(gamma[1]),
                     expression(beta), expression(sigma[y]),
                     expression(sigma[alpha]),
                     unlist(lapply(1:fake.n.counties,
                                   function(x) parse(text=paste('alpha[',x,']'))
                                   ))))
frame10 <{ data.frame( x = 1:nrow(params),
                      y = params$delta,
                      y.min.50 = params$lower.50 - params$actual,
                      y.max.50 =  params$upper.50 - params$actual,
                      y.min.95 = params$lower.95 - params$actual,
                      y.max.95 =  params$upper.95 - params$actual)
p10 <{ ggplot( frame10, aes(x = x, y = y) ) +
  scale_y_continuous("actual - fitted") +
  scale_x_discrete("parameter", labels=xlabs) +
  geom_pointrange(aes(ymin = y.min.95, ymax = y.max.95, y = y),
                  #                  linetype='dashed', size=I(1),
                  #                  color='darkslategray4') +
                  size=I(0.9), color='gray') +
  geom_pointrange(aes(ymin = y.min.50, ymax = y.max.50, y = y), size=I(0.9)) +
  geom_hline(yintercept=0)  +
  theme_bw() + theme(panel.grid=element_blank())

print(p10)
```

The code below produces Table 1:

```
# Table 1
mcc <{ function(n.col,text) paste('\\multicolumn{',n.col,'}{c}{',text,'}',
                                  sep='')
mr <{ function(n.row,text) paste('\\multirow{',n.row,'}{*}{',text,'}',sep='')
bf <{ function(text) paste('\\textbf{',text,'}',sep='')
fmt <{ function(x) format(round(x,2),x, digits=2,nsmall=2)
cbf <{ function(x) ifelse(abs(x)>1,paste('\\bsignif{',fmt(x),'}',
                                         sep=''),
                          paste('\\ensuremath{',fmt(x),'}',sep=''))
tex.rownames <{ data.frame(r = c('g0','g1','b','sigma.a','sigma.y'),
                           tex=c('\\gamma_0','\\gamma_1','\\beta',
                                 '\\sigma_\\alpha','\\sigma_y'),
                           stringsAsFactors=FALSE)
tex.rownames <{ rbind(tex.rownames, cbind(r = paste('a[',1:fake.n.counties,']',
                                                    sep=''),
                                          tex = paste('\\alpha_{',
                                                      1:fake.n.counties,'}',
                                                      sep='')))
pt <{ paste("\\begin{table}\n\\centering\n",
            "\\begin{tabular}{cS[table-format=3.2]S[table-format=3.2]",
            "S[table-format=3.2]S[table-format=3.2]S[table-format=3.2]",
            "S[table-format=3.2]S[table-format=3.2]",
            "S[table-format=2.2]S[table-format=2.2]}",sep='')
pt <{ paste(pt,'\\hline',sep='\n')
pt <{ paste(pt,'\\sisetup{detect-weight=true,detect-inline-weight=math}\n')
pt <{ paste(pt,paste(paste(mr(3,''),mcc(1,mr(3,bf('actual'))),
                           mcc(5,bf('Highest Density Interval (HDI)')),
                           mcc(1,mr(3,bf('actual $-$ fitted'))),
                           '','', sep=' & '),
                     '\\\\',sep=''),sep='\n')
pt <{ paste(pt,paste(paste('','',mcc(2,bf('lower')),mcc(1,mr(2,bf('mean'))),
                           mcc(2,bf('upper')),'',mcc(2,bf('scaled error')),
                           sep=' &'),
                     '\\\\',sep=''),sep='\n')
pt <{ paste(pt,paste(paste('','',mcc(1,bf('95\\%')),mcc(1,bf('50\\%')),'',
                           mcc(1,bf('50\\%')),mcc(1,bf('95\\%')),'',
                           mcc(1,bf('50\\%')), mcc(1,bf('95\\%')),sep=' &'),
                     '\\\\',sep=''),sep='\n')
pt <{ paste(pt,'\\hline',sep='\n')
for (i in 1:nrow(params)) {
  pt <{ paste( pt, paste( with(params[i,], paste( paste('$',
              tex.rownames$tex[tex.rownames$r == rownames(params)[i]], '$',
              sep=''),
              fmt(actual), fmt(lower.95),fmt(lower.50),fmt(fitted),
              fmt(upper.50),fmt(upper.95), fmt(delta),
              cbf(std.res.50),cbf(std.res.95), sep=' & ')),
              '\\\\',sep=''),sep='\n')
}
pt <{ paste(pt,'\\hline\n\\end{tabular}',sep='\n')
pt <{ paste(pt,paste("\\caption[Fake data parameters and fitted values.]",
         "{Fake data parameters and fitted values. The column ‘‘scaled error’’ ",
         "represents the ratio  of the error (actual $-$ fitted) to the HDI. ",
         "If the scaled error is greater than 1, then the actual value lies outside ",
         "the HDI, and is marked in boldface.}\\label{tab:fake.data}",sep=''),sep='\n')
pt <{ paste(pt,'\\end{table}',sep='\n')
asis_output(pt)
```

The code below produces Fig. 11, showing the posterior probability density of σ_a.

```
p11 <{ ggplot(as.data.frame(post.fake), aes(x=sigma.a)) +
  geom_histogram(binwidth=0.01, fill=I('gray')) +
  annotate('rect', xmin = fake.hdi.50.sigma.a['lower'],
           xmax = fake.hdi.50.sigma.a['upper'],
           ymin = 0, ymax=Inf, fill='dark red', alpha=0.2) +
  geom_vline(xintercept=mean(post.fake[,'sigma.a'])) +
  labs(x=expression(sigma[alpha])) + theme_bw()
print(p11)
```

References

Gelman, Andrew, & Jennifer Hill (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press. 654 pp.

Gelman, Andrew, *et al*. (2013). *Bayesian Data Analysis, Third Edition*. CRC Press. 677 pp.

Kruschke, John K. (2012). *Why to use highest density intervals instead of equal tailed intervals*. Doing Bayesian Data Analysis. url: `http://doingbayesiandataanalysis.blogspot.com/2012/04/why-to-use-highest-density-intervals.html` (visited on 07/17/2014).

Kruschke, John K. (2013a). \Bayesian estimation supersedes the t test". 142.2, pp. 573{603. doi: `10.1037/a0029146`.

Kruschke, John K. (2013b). *Decisions from posterior distributions: Tail probability or highest density interval?* Doing Bayesian Data Analysis. url: `http://doingbayesiandataanalysis.blogspot.com/2013/07/decisions-from-posterior-distributions.html` (visited on 07/17/2014).

Metropolis, Nicholas (1987). \The Beginning of the Monte Carlo Method". *Los Alamos Sceince* 15, pp. 125{130. url: `http://jackman.stanford.edu/mcmc/metropolis1.PDF`.

Price, Phillip N. (2014). *Everything I need to know about Bayesian statistics, I learned in eight schools*. Statistical Modeling, Causal Inference, and Social Science Blog. url: `http://andrewgelman.com/2014/01/21/everything-need-know-bayesian-statistics-learned-eight-schools/` (visited on 07/09/2014).

Price, Phillip N., Anthony V. Nero & Andrew Gelman (1996). \Bayesian prediction of mean indoor radon concentrations for Minnesota counties". *Health Physics* 71, pp. 922{936.