Basics and Issues with the BUGS Modeling Language

JEFF GILL
Department of Government
Department of Mathematics and Statistics
Center for Behavioral Neurosciences
American University

BUGS Software for MCMC

- A relatively new convenience: flexible, powerful, but sometimes fragile.
- The commands are relatively R-like, but unlike R there are relatively few functions to work with.
- Written in Pascal.
- Unlike other programming languages, statements are not processed serially, they constitute a full specification.
- Four steps to producing useful MCMC generated inferences in BUGS .
  1. Specify the distributional features of the model, and the quantities to be estimated.
  2. Compile the instructions into the run-time program.
  3. Run the sampler that produces Markov chains.
  4. Assess convergence using basic diagnostics in BUGS or the more sophisticated suites of programs, CODA and BOA that run under R.

Ways To Get the BUGS Language Software

- WinBUGS : The Bugs Project at: http://www.mrc-bsu.cam.ac.uk/bugs/.
- R2WinBUGS: CRAN package at: http://cran.r-project.org/web/packages/R2WinBUGS/index.html.
- JAGS : Just Another Gibbs Sampler at: http://www-fis.iarc.fr/~martyn/software/jags/.
- R2jags: CRAN package at: http://cran.r-project.org/web/packages/R2jags/index.html.
- Rjags: http://www.johnmyleswhite.com/notebook/2010/08/20/... ...using-jags-in-r-with-the-rjags-package/
- Running BUGS from Stata: http://www.personal.leeds.ac.uk/~hssdg/Stata/index.htm.
- openBUGS: http://mathstat.helsinki.fi/openbugs/.

Specifying Models with BUGS

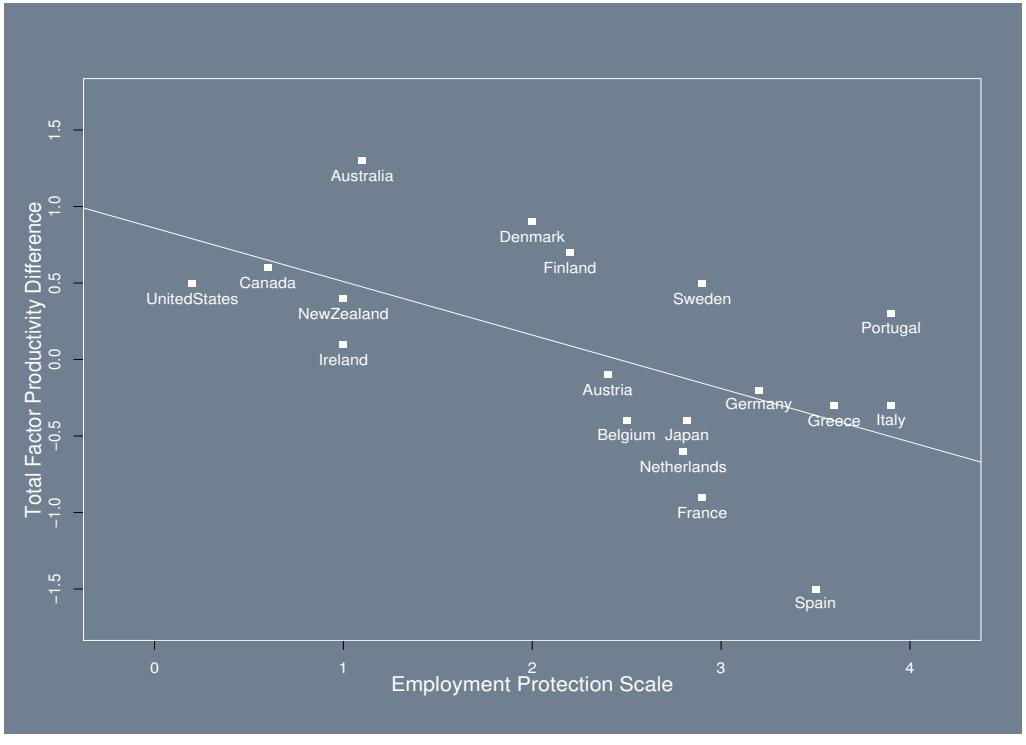
- The default MCMC transition kernel is the Gibbs sampler and therefore the first step must identify the full conditional distributions for each variable in the model.
- The second major component of this process is the iterative (human) cycling between the third and fourth steps.
- Recall that the Markov chain is conditionally dependent on just the last step, so the only information that needs to be saved in order to restart the chain is the last chain value produced.
- Clicking the `Coda` causes `WinBUGS` to write these values to a file named `*.out` with the index `*.ind`.
- Even if you use `JAGS` only you need the `R2WinBUGS` package if you want to use `write.model`.

Specifying Models with BUGS (cont.)

- BUGS Vocabulary:
  - **node**: values and variables in the model, specified by the researcher.
  - **parent**: a node with direct influence on other nodes.
  - **descendent**: the opposite of a parent node, but also can be a parent.
  - **constant**: a "founder node", they are fixed and have no parents.
  - **stochastic**: a node modelled as a random variable (parameters or data).
  - **deterministic**: logical consequences of other nodes.

Linear BUGS Example

- Consider the following economic data from Organization for Economic Cooperation and Development (OECD) that highlight the relationship between commitment to employment protection measured on an interval scale (0 to 4) indicating the quantity and extent of national legislation to protect jobs, and the total factor productivity difference in growth rates between 1980-1990 and 1990-1998 (see *The Economist*, September 23, 2000 for a discussion).



Linear BUGS Example (cont.)

```
list(x= c(0.20, 0.60, 1.10, 1.00, 1.00, 2.00, 2.20, 2.40, 2.50, 2.82, 2.90,
              2.80, 2.90, 3.20,3.60, 3.90, 3.90, 3.50),
      y= c(0.5,  0.6,  1.3,  0.4,  0.1,  0.9,  0.7, -0.1, -0.4, -0.4,  0.5, -0.6,
             -0.9, -0.2, -0.3,  0.3, -0.3, -1.5),  N=18)

list(alpha = 0.0, beta = 0.0, tau = 1.0)
```

Linear BUGS Example (cont.)

- We know from Gauss-Markov theory that the posterior distribution of both the intercept and the slope coefficients is student's-t with $n - k - 1 = 17$ degrees of freedom.
- So why are we running BUGS on a linear model? Consider how different the estimation process really is here:

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

versus

$$\begin{aligned} \alpha_1 &\sim f(\alpha|\beta_0), & \beta_1 &\sim f(\beta|\alpha_1) \\ \alpha_2 &\sim f(\alpha|\beta_1), & \beta_2 &\sim f(\beta|\alpha_2) \\ &\vdots & &\vdots \\ \alpha_m &\sim f(\alpha|\beta_{m-1}), & \beta_m &\sim f(\beta|\alpha_m). \end{aligned}$$

not to mention *priors*!

Linear BUGS Example (cont.)

- First define the *statistical structure* of the model:

```
mu[i] <- alpha + beta*x[i];
y[i] ~ dnorm(mu[i],tau)
```

  Note that we are indexing across the data here (not chaining!).

- Now define the variables in the model and their distributional assumptions:

```
alpha ~ dnorm(0.0,0.001)
beta ~ dnorm(0.0,0.001)
tau ~ dgamma(1,0.1)
```

  The second normal parameter is a *precision* not a variance, by convention.

Linear BUGS Example (cont.)

- Full Model:

```
model {
    for (i in 1:n) {
        mu[i] <- alpha + beta*x[i]
        y[i] ~ dnorm(mu[i],tau)
    }
    alpha ~ dnorm(0.0,0.001)
    beta  ~ dnorm(0.0,0.001)
    tau   ~ dgamma(1,0.1)
}
```

- Run for 50,000 iterations of the Markov chain.
- Using the posterior mean as a point estimate, we can compare with `lm`:

	OLS Model		MCMC Posterior	
	Estimate	Std. Error	Mean	Std. Error
(Intercept)	0.859	0.317	0.859	0.322
Slope	-0.349	0.121	-0.349	0.123

- Observations?

Linear BUGS Example (cont.)

- The following general steps are given for a JAGS implementation where each command corresponds to a specific button in WinBUGS .

```
model in "oecd.bug"
data in "oecd-data.R"
compile
inits in "oecd-init.R"
initialize
update 1000
monitor set alpha
monitor set beta
monitor set tau
update 5000
coda *
exit
```

Running the Linear BUGS Example

```
lapply(c("rjags","R2jags","arm","coda","superdiag","R2WinBUGS"),library,
    character.only=TRUE)
oecd <- read.table("http://jeffgill.org/files/jeffgill/files/oecd.dat__0.txt",
    header=TRUE)
oecd.list <- list("Prot"=oecd$Prot,"Prod"=oecd$Prod,n=18)
oecd.mod <- function() {
    for (i in 1:n) {
        mu[i] <- alpha + Prot[i]
        Prod[i] ~ dnorm(mu[i],tau)
    }
    alpha ~ dnorm(0.0,0.001)
    beta  ~ dnorm(0.0,0.001)
    tau   ~ dgamma(1,0.1)
}
oecd.params <- c("alpha","beta","tau")
oecd.out <- jags(data=oecd.list, parameters.to.save=oecd.params,
    n.iter=5000, model=oecd.mod,n.burnin=2500, n.thin=1,n.chains=1)
```

Outpout for the Linear BUGS Example

```
update(oecd.out, n.iter=5000, n.burnin=0, n.thin=1,n.chains=1)
  |**************************************************| 100%
Inference for Bugs model at "/var/...", fit using jags,
 3 chains, each with 2000 iterations (first 0 discarded)
 n.sims = 6000 iterations saved
          mu.vect sd.vect    2.5%     25%    50%    75%  97.5%  Rhat
alpha      -2.328   0.390  -3.084  -2.592 -2.326 -2.066 -1.559 1.001
beta       -0.538  31.445 -62.055 -21.838 -0.276 20.806 60.743 1.001
tau         0.408   0.131   0.191   0.314  0.392  0.489  0.693 1.001
deviance   70.056   1.929  68.137  68.680 69.458 70.825 75.230 1.001
          n.eff
alpha      6000
beta       6000
tau        6000
deviance   6000
```

Outpout for the Linear BUGS Example

```
oecd.mcmc <- as.mcmc(oecd.out)
oecd.mat <- oecd.mcmc[[1]]
names(oecd.mat) <- c("alpha","beta","deviance","tau")
head(oecd.mat)

Markov Chain Monte Carlo (MCMC) output:
Start = 2501
End = 2507
Thinning interval = 1
         alpha       beta deviance       tau
[1,] -2.350854 -24.648258 68.11275 0.3686066
[2,] -2.325837 -18.471335 72.26484 0.7171788
[3,] -1.773312 -18.429619 71.55390 0.5044848
[4,] -1.869802  -7.707788 70.21213 0.2676663
[5,] -1.813388 -87.277033 71.69755 0.5368300
[6,] -1.797037  19.724388 69.96334 0.3471874
[7,] -1.987507  -5.222878 68.91109 0.3512523
```

Exercise 12: OECD Data

- Rerun the OECD linear example with different priors.
- Compare your results to a regular `lm()` model that you run.
- How much do your priors matter?
- Code below to get you started.

Healthcare Example

- The Study: 1180 children in Florida who visited their HMO clinic and did or did not require an emergency room visit shortly thereafter.
- The data:
  - *erodd*, the dichotomous outcome variable, [0,1], indicating whether or not there was an emergency room visit.
  - *metq*, a severity score, [1,2,3], indicating the degree of illness diagnosed at the HMO visit.
  - *np*, indication of profit, [1], or nonprofit, [-1], status of the HMO. This is the key explanatory variable of interest.

Healthcare Example

- The Model:

$$
\begin{aligned}
erodd_i &\sim \mathcal{BE}(p_i) \\
\text{logit}(p_i) &= \alpha_0 + \alpha_1 metq_i + \alpha_2 np_i \\
\alpha_0 &\sim \mathcal{N}(0, 10) \\
\alpha_1 &\sim \mathcal{N}(0, 10) \\
\alpha_2 &\sim \mathcal{N}(0, 10)
\end{aligned}
$$

- Where the results of interest are the posterior distributions of α_0, α_1, and α_2.
- The model is run with a burn-in period of 1,000 cycles and the run for 11,000 more.

Healthcare Example

- The Code:

```
model
{
    for( i in 1 : N ) {
        logit(p[i]) <- alpha0 + alpha1 * metq[i] + alpha2 * np[i]
        erodd[i] ~ dbern(p[i])
    }
    alpha0 ~ dnorm(0.0,0.1)
    alpha1 ~ dnorm(0.0,0.1)
    alpha2 ~ dnorm(0.0,0.1)
}

list(alpha0 = 0, alpha1 = 0, alpha2 = 0)

list(erodd = c(1,1,1,0,1,0,1,...),metq = c(3,3,3,3,3,2,3,...),
     np = c(-1,-1,-1,1,-1,-1,1,...),N=1180)
```

Healthcare Example

- The Results:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
alpha0	-1.971	0.22280	0.008254	-2.41300	-1.968	-1.540	1	11000
alpha1	0.2808	0.09423	0.003505	0.098070	0.2803	0.4645	1	11000
alpha2	0.1646	0.08042	9.194E-4	0.008893	0.1639	0.3213	1	11000

Thermonuclear Testing Example

- There were certainly many political, economic, engineering, and policy reasons for conducting nuclear tests in the United States, we will specify a model with only three predictors for each year: U.S. gross domestic product (in trillions of dollars, base 1996), U.S domestic non-financial debt (in trillions of dollars, base 1996, outstanding credit market debt of federal, state, and local government as well as private non-financial sectors), and counts of Soviet tests.
- Specify a simple Bayesian generalized linear model with a Poisson link function according to:

```
model {
    for (i in 1:n)  {
        log(mu[i]) <- b0 + b1*US.GDP[i] + b2*US.DEBT[i]
                         + b3*(SOV.TEST[i]-mean(SOV.TEST))
        US.TEST[i] ~ dpois(mu[i])
    }

    b0 ~ dnorm(0,1.0E-4)
    b1 ~ dnorm(0,1.0E-4)
    b2 ~ dnorm(0,1.0E-4)
    b3 ~ dunif(-100,100)
}
```

Thermonuclear Testing Example

- JAGS commands (I usually cut-and-paste from a file):

```
model in "nukes2.bug"
data in "nukes2.data"
compile
inits in "nukes2.init"
initialize
update 500000
monitor set b0
monitor set b1
monitor set b2
monitor set b3
update 1000000
coda *
exit
```

Thermonuclear Testing Example, Summary of Results

			Posterior Quantiles				
	Mean	SD	2.5%	25%	50%	75%	97.5%
CONSTANT	3.835	0.078	3.683	3.783	3.836	3.888	3.988
US.GDP	-0.903	0.250	-1.398	-1.073	-0.902	-0.734	-0.412
US.DEBT	0.345	0.131	0.088	0.257	0.345	0.434	0.603
SOV.TEST	0.017	0.005	0.008	0.014	0.017	0.020	0.026

Running JAGS : Blood Pressure Example, Data Setup

```
pressure <- c(132,155,130,142,150,128,126,118,180,124,150,134,140,142,128)
age.years <- c(49,56,52,46,57,42,43,45,56,52,42,57,56,56,53)
weight.lb <- c(145,216,115,170,172,166,164,152,275,221,175,132,188,178,168)
n <- 15
```

Turning into a list...

```
bp.list <- list("pressure"=pressure,"age.years"=age.years,"weight.lb"=weight.lb,"n"=15)
```

Running JAGS : Blood Pressure Example, Model Definiition in R

```
bp.model <- function() {
    for (i in 1:n) {
        mu[i] <- alpha + beta1*age.years[i] + beta2*weight.lb[i];
        pressure[i] ~ dnorm(mu[i],tau)
    }
    alpha  ~  dnorm(0.0,0.00001)
    beta1  ~  dnorm(0.0,0.00001)
    beta2  ~  dnorm(0.0,0.00001)
    tau    ~  dgamma(2.0,0.1)
}
```

Running JAGS : Blood Pressure Example in a JAGS Window

```
model in "blood.pressure.jags"
data in "blood.pressure.jags.dat"
compile
inits in "blood.pressure.jags.init"
initialize
update 200000
monitor set alpha
monitor set beta1
monitor set beta2
monitor set tau
update 200000
coda *
exit
```

Running jags: Blood Pressure Example

```
. Reading data file blood.pressure.jags.dat
. Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 93
. Reading initial values file blood.pressure.jags.init
. . Updating 200000
-------------------------------------| 200000
************************************* 100%
. Updating 200000
-------------------------------------| 200000
************************************* 100%
```

Running JAGS : Blood Pressure Example

```
library(coda)
bp <- read.coda("CODAchain1.txt","/Users/jgill/Bugs/CODAindex.txt")
apply(bp,2,summary)
        alpha  beta1  beta2     tau
Min.    -61.1 -2.980 -0.354 0.00109
1st Qu.  44.7  0.322  0.198 0.00724
Median   61.8  0.654  0.248 0.00933
Mean     61.9  0.654  0.247 0.00973
3rd Qu.  78.6  0.993  0.298 0.01180
Max.    218.0  3.150  0.722 0.03550
```

Running JAGS from R

```
lapply(c("rjags","R2jags","arm","coda","superdiag","R2WinBUGS"),library,
    character.only=TRUE)
bp.params <- c("alpha","beta1","beta2","tau")
bp.out <- jags(data=bp.list, parameters.to.save=bp.params,
    n.iter=5000, model=bp.model,n.burnin=2500, n.thin=1,n.chains=5)
update(oecd.out, n.iter=5000, n.burnin=0, n.thin=1,n.chains=1)
  |**************************************************| 100%
 1 chains, each with 5000 iterations (first 0 discarded)
 n.sims = 5000 iterations saved
         mu.vect sd.vect    2.5%     25%    50%    75%  97.5%
alpha     -2.328   0.392  -3.101  -2.588 -2.329 -2.075 -1.544
beta       0.794  31.487 -61.074 -20.559  1.014 22.185 61.342
tau        0.410   0.135   0.190   0.313  0.395  0.490  0.717
deviance  70.112   2.083  68.145  68.681 69.440 70.822 75.808

DIC info (using the rule, pD = var(deviance)/2)
pD = 2.2 and DIC = 72.3
DIC is an estimate of expected predictive error (lower deviance is better).
```

Running JAGS from R

```
bp.mcmc <- as.mcmc(bp.out)
bp.mat <- bp.mcmc[[1]]
head(bp.mat,3)
Markov Chain Monte Carlo (MCMC) output:
Start = 2501
End = 2507
Thinning interval = 1
        alpha     beta1     beta2 deviance         tau
[1,] 60.95750 0.6641578 0.2497573 114.0996 0.006094437
[2,] 27.56683 1.5017660 0.2079501 117.2176 0.004862224
[3,] 35.05893 0.8142167 0.3615621 115.8884 0.007580338
```

Ordered Logit Example

- The 1960 U.S. presidential election between John Kennedy and Richard Nixon was one of the closest contests in national history, with Kennedy's margin of victory less than one percent of the popular vote.
- Campaigns, journalistic accounts, and social contexts can modify pre-election perceptions of the competitiveness of an uncertain outcome.
- This has implications for turnout and therefore may affect the election as well.
- This example uses the 1960 American National Election Study to explore the link between personal characteristics and perception of closeness of the impending election.
- The outcome variable has four ordered categories:
  - ▷ one candidate will win by a lot,
  - ▷ one candidate will win by quite a bit,
  - ▷ this will be a close race–fairly even,
  - ▷ this will be a very close race.

Ordered Logit Example

- The specified explanatory variables are:
  - `education`, 1=8th grade or lower (233 cases), 2=highschool (428 cases), 3=some college or more (185 cases).
  - `sex`, 1=male, (412 cases), 2=female (434 cases).
  - `seedebates`, 1=no (141 cases), 2=yes (660 cases).
  - `importance`, 1=care very much (270 cases), 2=care pretty much (308 cases), 3=pro-con/depends (6 cases), 4=don't care very much (155 cases), 5=don't care at all (85 cases).
  - `involvement`, 8 categories from low to high with the distribution of cases: $(158, 165, 245, 101, 78, 55, 4, 39)$.
  - `catholic`, 1=no (623 cases), 2=yes (179 cases).
  - `partyid`, 1=strong Democrat (198 cases), 2=not very strong Democrat (201 cases), 3=independent closer to Democrats (52 cases), 4=independent (67 cases), 5=independent closer to Republicans (59 cases), 6=not very strong Republican (117 cases), 7=strong Republican (139 cases).
- Note that there are no 0 codings.

Ordered Logit Example

- Ordered choice models are constructed by assuming that there is a continuous latent metric dictating the categorical choices since researchers construct the scale not the respondents.
- So the outcome variable $\mathbf{Y}$ has C ordered categories separated by estimated thresholds (sometimes called "cutpoints" or "fences") sitting over a continuous utility metric $\mathbf{U}$ that cannot be seen:

$$\mathbf{U}_i : \ \theta_0 \underset{c=1}{\Longleftrightarrow} \theta_1 \underset{c=2}{\Longleftrightarrow} \theta_2 \underset{c=3}{\Longleftrightarrow} \theta_3 \ldots \theta_{C-1} \underset{c=C}{\Longleftrightarrow} \theta_C,$$

  where the end-categories extend out to $-\infty$ and ∞, respectively.
- The effect of the explanatory variables is determined by a linear additive specification on the latent scale such that the ith person's utility is $U_i = \mathbf{X}_i\boldsymbol{\beta} + \epsilon_i$, where the $\boldsymbol{\beta}$ do not depend on the θ values.
- Some authors prefer a minus sign in front of $\mathbf{X}_i\boldsymbol{\beta}$, but the model defined here does not as is also the case with the `R` function `polr`.

Ordered Logit Example

- The vector of utilities across individuals is determined in the following way:
  - the probability of the ith person choosing the kth category or less is $p(Y_i \leq k|\mathbf{X})$,
  - this is equal to the probability that the ith person's utility is less than or equal to next threshold to the right of this category, $p(U_i \leq \theta_k)$,
  - now substitute in the linear additive component for the utility to get $p(\mathbf{X}_i\boldsymbol{\beta} + \epsilon_i \leq \theta_k)$,
  - rearrange to leave the stochastic component alone on the left $p(\epsilon_i \leq \theta_k - \mathbf{X}_i\boldsymbol{\beta})$,
  - notice that this is just the CDF of ϵ_i, $F_{\epsilon_i}(\theta_k - \mathbf{X}_i\boldsymbol{\beta})$, at the point $\theta_k - \mathbf{X}_i\boldsymbol{\beta}$,
  - specifying a logistic distribution for this distribution results in the model $p(Y_i \leq k|\mathbf{X}) = [1 + \exp(\mathbf{X}_i\boldsymbol{\beta} - \theta_k)]^{-1}$.
- An ordered probit model can also be created by specifying $p(Y_i \leq k|\mathbf{X}) = \Phi(\theta_k - \mathbf{X}_i\boldsymbol{\beta})$ instead.
- Reminder about `R`: $logitP(Y <= k|x) = zeta_k - eta$ ($\eta = \mathbf{X}\boldsymbol{\beta}$).

Ordered Logit Example

- Specify two important datastructures:
  - ▷ `Q[i,j]`, the cumulative probability that the ith case is in the jth category or less.
  - ▷ `p[i,j]`, the marginal probability that the ith case is in the jth category.
- The logistic specification is specified by `for (j in 1:(Ncat-1)) logit(Q[i,j]) <- cut[j] - mu[i]` , which loops through each of the first $k-1$ categories relating the linear additive component to a cumulative probability.
- The last cumulative category is not necessary to calculate since it equals one.
- Next we want to calculate the marginal probabilities from the cumulative probabilities.
- The first category is easy since they are equivalent: `p[i,1] <- Q[i,1]`.
- Then we loop through all of the higher categories except for the last, differencing the adjacent cumulative probabilities to get the marginal probabilities: `for (j in 2:(Ncat-1)) p[i,j] <- Q[i,j] - Q[i,(j-1)]` .
- Finally the right-most category is obtained by `p[i,Ncat] <- 1 - Q[i,(Ncat-1)]`.

Ordered Logit Example

- The outcome variable is then modeled with these probabilities with: `close[i]` $\sim$ `dcat(p[i,1:Ncat])`.
- The last line in the `1:N` loop creates a residual vector, which is then summarized outside of the loop with a standard error function.
- Priors:
  - $\boldsymbol{\beta}$ coefficients and the cutpoints are given Cauchy priors with 5 degrees of freedom, centered at zero.
  - The last line creates the variable `cut` from the intermediate variable `cut0` to create a sorted form from the prior distribution.

Ordered Logit Example, Code

```
model {
    for (i in 1:Nsub) {
        mu[i] <- beta[1]*education[i] + beta[2]*sex[i]
            + beta[3]*seedebates[i] + beta[4]*importance[i]
            + beta[5]*involvement[i]
            + beta[6]*importance[i]*involvement[i]
            + beta[7]*catholic[i] + beta[8]*partyid[i]
        for (j in 1:(Ncat-1)) { logit(Q[i,j]) <- cut[j] - mu[i] }
        p[i,1] <- Q[i,1]
        for (j in 2:(Ncat-1)) { p[i,j] <- Q[i,j] - Q[i,(j-1)] }
        p[i,Ncat] <- 1 - Q[i,(Ncat-1)]
        close[i] ~ dcat(p[i,1:Ncat])
        E.y[i] <- close[i] - mu[i]
    }
    sd.y <- sd(E.y[])
    for (k in 1:Nvar) { beta[k] ~  dt(0,1,5) }
    for (k in 1:(Ncat-1)) {  cut0[k] ~  dt(0,1,5) }
    cut[1:(Ncat-1)] <- sort(cut0)
}
```

Ordered Logit Example, Results

Parameters	Mean	Std. Error	95% HPD Interval
`education`	0.2773	0.1067	[0.0681: 0.4864]
`sex`	0.3708	0.1410	[0.0944: 0.6472]
`seedebates`	0.4438	0.1813	[0.0884: 0.7992]
`importance`	-0.0432	0.1235	[-0.2852: 0.1988]
`involvement`	-0.3021	0.1045	[-0.5070:-0.0973]
`importance`×`involvement`	0.0534	0.0280	[-0.0015: 0.1084]
`catholic`	0.2484	0.1741	[-0.0928: 0.5896]
`partyid`	0.0403	0.0328	[-0.0240: 0.1047]
θ_1	-1.7139	0.5141	[-2.7215:-0.7064]
θ_2	-0.2879	0.4989	[-1.2658: 0.6900]
θ_2	3.0812	0.5142	[2.0734: 4.0890]

$s_y = 0.7517$, Model DIC: 1580.64, Null DIC: 1609.95

JAGS Versus BUGS on truncation and censoring

- JAGS uses the T(,) construct for truncation where as BUGS uses I(,)..
- Censoring in JAGS is represented by the novel distribution dinterval.
- First define an indicator vector for the variable, say is.censored, where for each value is.censored[i]=0 if it is noncensored and is.censored[i]=1 if it is censored.
- The variable itself, say X1, contains data values for the noncensored and NA for the censored.
- These are related in the model by: is.censored ~ dinterval(X1,censor.limit) for a single censoring point, or is.censored ~ dinterval(X1,censor.limit[i]) for variable censoring points.
- Then JAGS imputes a random value for X1 from the likelihood function you specify.
- From the JAGS manual, an example of right-hand failure time censoring in a loop:

```
is.censored[i] ~ dinterval(t[i], t.censor[i])
t[i] ~ dweib(r,mu[i])
```

where: t.censor[i] is a censoring time, is.censored[i] is a censoring indicator for the ith case, t[i] is a failure time.

Left Censoring Example In JAGS

```
bounds.data                  L        U
                [1,] 14.98266 15.68029
                [2,] NA       21.91590
                [3,] 18.34953 19.04716....

censor.model <- function() {
    for (i in 1:50) {
        is.censored[i] ~ dinterval(t[i], bounds.data[i,])
        t[i] ~ dnorm(mu,tau)
    }
    mu ~ dnorm(0,0.01)
    tau ~ dgamma(1,0.01)
}

write.model(censor.model,local.model); data <- list("bounds.data"=bounds.data)
inits <- list(mu=rnorm(1),tau=rgamma(1,1,.01),t=as.vector(apply(bounds.data,1,mean)))
censor.out <- jags(data,inits, c("mu","tau"), model.file="local.model", n.chains=5,
                   n.iter=50000,n.burnin=10000, n.thin=1, DIC=TRUE,
                   working.directory=NULL,refresh = 50000/50, progress.bar = "text")
```

A Hierarchical Model of Lobbying Influence in the US States

- The American State Administrator's Project (ASAP) survey asks administrators about the influence of a variety of external political actors including "clientele groups" in their agencies.
- Consider a 713×22 matrix $\mathbf{X}$ with a leading column of 1's for individual level explanatory variables, and a 50×3 matrix $\mathbf{Z}$ for state-level explanatory variables.
- Our outcome variable (`group influence`) measures the respondents' perception of interest groups' influence on total budget, special budgets, and general public policies.
- We relate these variables through the linear hierarchical model:

$$\begin{aligned} Y_i &\sim N(\boldsymbol{\alpha}_{j[i]} + \boldsymbol{\beta}\mathbf{X}_i, \sigma_y^2), \qquad \text{for } i = 1, \ldots, 713 \\ \boldsymbol{\alpha}_j &\sim N(\boldsymbol{\gamma}\mathbf{Z}, \sigma_\alpha^2), \qquad \text{for } j = 1, \ldots, 50, \end{aligned}$$

  using hierarchical notation from Gelman and Hill (2007) to indicate that the ith respondent is nested in the jth state: $\boldsymbol{\alpha}_{j[i]}$ to get a state-specific random intercept.
- This random intercept is the parameterized at the second level by the three explanatory variables in $\mathbf{Z}$ and their corresponding estimated coefficients, $\boldsymbol{\gamma}$.

A Hierarchical Model of Lobbying Influence in the US States

- Note the two different variances accounted for in this specification. The term σ_y^2 measures the *within-state* variance of the outcome variables, whereas the term σ_α^2 gives the variance of the mean estimates *between-states*.

- Prior distributions are diffuse normals centered at the point estimates from Kelleher and Yackee (2009), Model 3 (page 593), for overlapping covariate use, but diffuse for others.

- We deviate from their values in only two ways: we substitute the variable `elected/board` for their variable Merit Position, and since we are using 2008 data only there cannot be a dummy variable for the year 2004.

- Thus:
$$\boldsymbol{\beta} \sim N(\boldsymbol{\beta}_{ky}, \boldsymbol{\Sigma}_{\boldsymbol{\beta}})$$
$$\boldsymbol{\gamma} \sim N(\boldsymbol{\gamma}_{ky}, \boldsymbol{\Sigma}_{\boldsymbol{\gamma}}),$$
where the $\boldsymbol{\Sigma}_{\boldsymbol{\beta}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\gamma}}$ matrices are diagonal forms with large variances relative to the Kelleher and Yackee point estimates.

- Code at (October 10) from
`http://jeffgill.org/classes/american-university-statistics-618spa-696-every-fall-bayesian-statistics-social-and`

A Hierarchical Model of Lobbying Influence in the US States, JAGS Code

```
model  {
    for (i in 1:SUBJECTS) {
        mu[i] <- alpha[state.id[i]]
            + beta[1]*contractind[i] + beta[2]*govinf3[i] + beta[3]*leginf3[i]
            + beta[4]*clientappt[i] + beta[5]*years[i] + beta[6]*gender[i]
            + beta[7]*educ[i] + beta[8]*party5pt[i] + beta[9]*X_Ifuncat13_2[i]
            + beta[10]*X_Ifuncat13_3[i] + beta[11]*X_Ifuncat13_4[i]
            + beta[12]*X_Ifuncat13_5[i] + beta[13]*X_Ifuncat13_6[i]
            + beta[14]*X_Ifuncat13_7[i] + beta[15]*X_Ifuncat13_8[i]
            + beta[16]*X_Ifuncat13_9[i] + beta[17]*X_Ifuncat13_10[i]
            + beta[18]*X_Ifuncat13_11[i] + beta[19]*X_Ifuncat13_12[i]
            + beta[20]*timegroupsmed[i] + beta[21]*timemedXcont[i]
        groupinf3[i] ~ dnorm(mu[i],tau)
    }
    for (j in 1:STATES) {
        eta[j] <- gamma[1]*gov6099all[j] + gamma[2]*totreg[j]
                + gamma[3]*nonprofs.per10k[j]
        alpha[j] ~ dnorm(eta[j],tau.alpha)
    }
```

A Hierarchical Model of Lobbying Influence in the US States, JAGS Code

```
    beta[1]   ~ dnorm(0.070,1)     # PRIOR MEANS FROM KELLEHER AND YACKEE 2009, MODEL 3
    beta[2]   ~ dnorm(-0.054,1)
    beta[3]   ~ dnorm(0.139,1)
    beta[4]   ~ dnorm(0.468,1)
    beta[5]   ~ dnorm(0.017,1)
    beta[6]   ~ dnorm(0.207,1)
    beta[7]   ~ dnorm(0.056,1)
    beta[8]   ~ dnorm(0.039,1)
    beta[9]   ~ dnorm(0.0,1)
    beta[10]  ~ dnorm(0.0,1)
    beta[11]  ~ dnorm(0.0,1)
    beta[12]  ~ dnorm(0.0,1)
    beta[13]  ~ dnorm(0.0,1)
    beta[14]  ~ dnorm(0.0,1)
    beta[15]  ~ dnorm(0.0,1)
    beta[16]  ~ dnorm(0.0,1)
    beta[17]  ~ dnorm(0.0,1)
    beta[18]  ~ dnorm(0.0,1)
    beta[19]  ~ dnorm(0.0,1)
    beta[20]  ~ dnorm(0.184,1)
    beta[21]  ~ dnorm(0.146,1)
    gamma[1]  ~ dnorm(0.0,1)
    gamma[2]  ~ dnorm(0.0,1)
    gamma[3]  ~ dnorm(0.0,1)
    tau       ~ dgamma(1.0,1)
    tau.alpha ~ dgamma(1.0,1)
}
```

A Hierarchical Model of Lobbying Influence in the US States

- Using `beta[ ]` is more efficient for running the model and importing the samples.
- If you have the same priors for these coefficients, then the priors can be specified in a loop:

```
for (k in 1:21) beta[k] ~ dnorm(0,1)
```

- `JAGS` specifies starting values in `R` vector and scalar forms:

```
tau <- 10
tau.alpha <- 10
alpha <- c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
zeta <- 1
beta <- c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
gamma <- c(1,1,1)
```

A Hierarchical Model of Lobbying Influence in the US States, Data File

```
asap.jags.list <- list(
state.id <-
c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6,
:
2.9545, 9.9194, 0.8706, 0.5967, 3.9635, 3.5073, 1.1225, 3.4171,
0.4537),
STATES <- 50, SUBJECTS <- 713)
```

Using `rjags` from `R`, Data Handling (same model)

```
lapply(c("rjags","R2jags","arm","coda","superdiag","R2WinBUGS"),library,
       character.only=TRUE)

source("Article.JPART/asap.rjags.dat")  # READS IN THE LIST VERSION OF THE DATA

names(asap.jags.list) <- c("state.id","contracting","gov.influence","leg.influence",
  "elect.board","years.tenure","education","party.ID","category2","category3",
  "category4","category5","category6","category7", "category8","category9",
  "category10","category11","category12","med.time","medt.contr","grp.influence",
  "gov.ideology","lobbyists","nonprofits","STATES","SUBJECTS")
```

A Hierarchical Model of Lobbying Influence in the US States, Data in R

```
asap.jags.list
$state.id
 [1]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
[21]  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2  2
 :
[37]  2.1932  6.6298  0.7195  2.1484  0.6801  2.9545  9.9194  0.8706  0.5967
[46]  3.9635  3.5073  1.1225  3.4171  0.4537

$STATES
[1] 50

$SUBJECTS
[1] 713
```

Using `rjags` from `R` (same model)

```
# DEFINE THE MODEL
asap.model2.rjags  <- function()  {
for (i in 1:SUBJECTS) {
    mu[i] <- alpha[state.id[i]]
      + beta[1]*contracting[i] + beta[2]*gov.influence[i] + beta[3]*leg.influence[i]
      + beta[4]*elect.board[i] + beta[5]*years.tenure[i]  + beta[6]*education[i]
      + beta[7]*party.ID[i]    + beta[8]*category2[i]     + beta[9]*category3[i]
      + beta[10]*category4[i]  + beta[11]*category5[i]    + beta[12]*category6[i]
      + beta[13]*category7[i]  + beta[14]*category8[i]    + beta[15]*category9[i]
      + beta[16]*category10[i] + beta[17]*category11[i]   + beta[18]*category12[i]
      + beta[19]*med.time[i]   + beta[20]*medt.contr[i]
      grp.influence[i] ~ dnorm(mu[i],tau.y)
    }
    for (j in 1:STATES) {
      eta[j]  <- gamma[1]*gov.ideology[j]+gamma[2]*lobbyists[j]+gamma[3]*nonprofits[j]
      alpha[j] ~ dnorm(eta[j],tau.alpha)
    }
    beta[1]   ~ dnorm(0.070,1)  # PRIOR MEANS FROM KELLEHER AND YACKEE 2009, MODEL 3
    beta[2]   ~ dnorm(-0.054,1)
    beta[3]   ~ dnorm(0.139,1)
    beta[4]   ~ dnorm(0.051,1)
    beta[5]   ~ dnorm(0.017,1)
    beta[6]   ~ dnorm(0.056,1)
    beta[7]   ~ dnorm(0.039,1)
    beta[8]   ~ dnorm(0.0,1)    # DIFFUSE PRIORS
    beta[9]   ~ dnorm(0.0,1)
    beta[10]  ~ dnorm(0.0,1)
    beta[11]  ~ dnorm(0.0,1)
    beta[12]  ~ dnorm(0.0,1)
    beta[13]  ~ dnorm(0.0,1)
    beta[14]  ~ dnorm(0.0,1)
```

```
    beta[15]  ~ dnorm(0.0,1)
    beta[16]  ~ dnorm(0.0,1)
    beta[17]  ~ dnorm(0.0,1)
    beta[18]  ~ dnorm(0.0,1)
    beta[19]  ~ dnorm(0.184,1)  # PRIOR MEANS FROM KELLEHER AND YACKEE 2009, MODEL 3
    beta[20]  ~ dnorm(0.156,1)
    gamma[1]  ~ dnorm(0.0,1)    # DIFFUSE PRIORS
    gamma[2]  ~ dnorm(0.0,1)
    gamma[3]  ~ dnorm(0.0,1)
    tau.y     ~ dgamma(1.0,1)
    tau.alpha ~ dgamma(1.0,1)
}
```

Using `rjags` from `R` (same model)

```
# SETUP INITIAL VALUES AND PARAMETER NAMES
asap.inits <- function(){ list("tau.y" = 10, "tau.alpha" = 10,
    "beta" = rep(1,20), "gamma" = c(1,1,1)) }
asap.params <- c("beta","gamma","tau.y","tau.alpha")

# RUN THE SAMPLER AND COLLECT coda SAMPLES
asap.out <- jags(data=asap.jags.list, inits=asap.inits, asap.params, n.iter=5000,
    model="Article.JPART/asap.model2.rjags", DIC=TRUE)

# USE THE autojags() FUNCTION TO RUN UNTIL G&R < 1.1
#asap.out2 <- autojags(asap.out)

# OR update() TO RUN LONGER
asap.out3 <- update(asap.out, n.iter=200000)
asap.mcmc3 <- mcmc(asap.out3)

# GET SUMMARY OF SAMPLES
summary(asap3.mcmc)
```

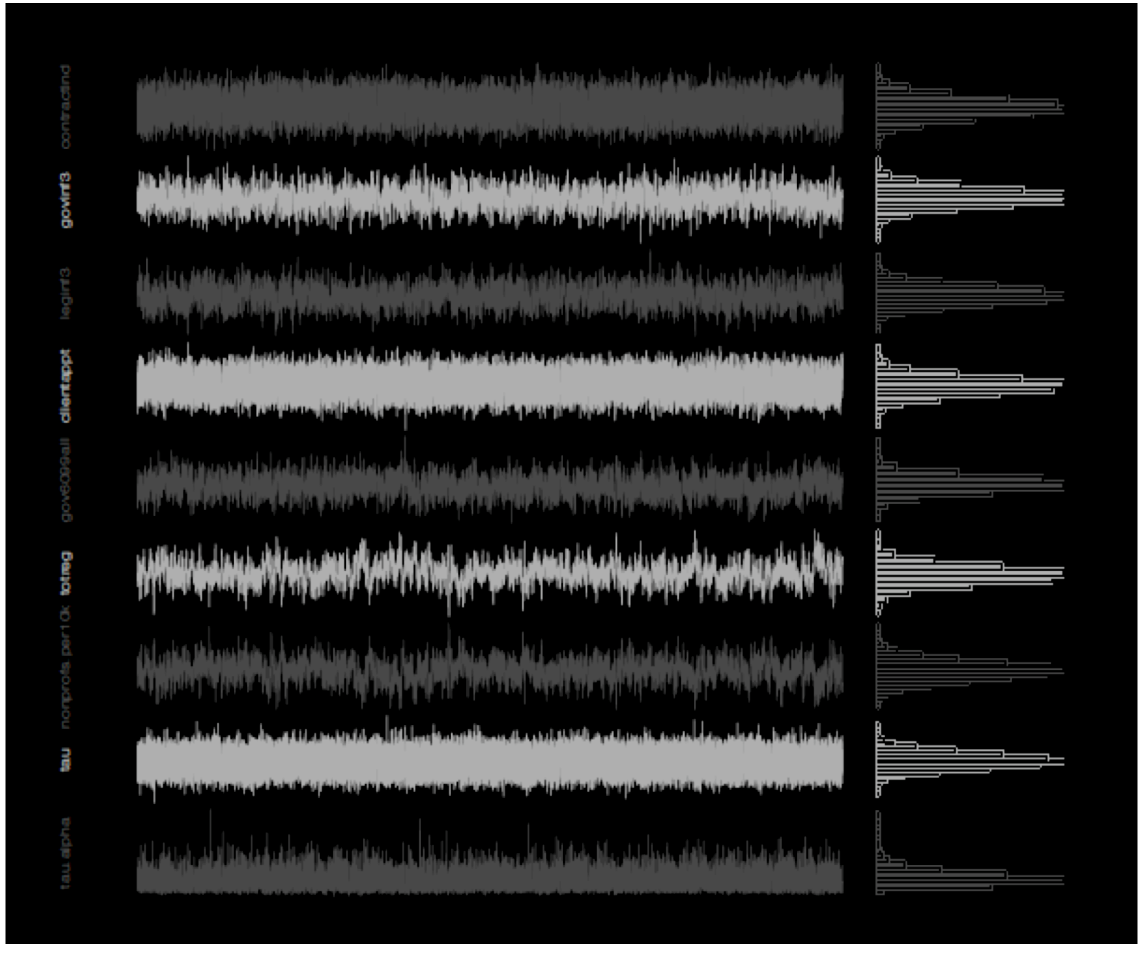
A Hierarchical Model of Lobbying Influence in the US States, Results

Parameters	Post.Mean	Post.SE	95%.Lower.CI	95%.Upper.CI
α mean(1:50)	1.4502	0.7076	0.0632	2.8372
`contractind`	0.1411	0.0966	-0.0482	0.3304
`govinf3`	0.0416	0.0379	-0.0327	0.1159
`leginf3`	0.3507	0.0403	0.2717	0.4296
`clientappt`	1.2318	0.3515	0.5428	1.9207
`years`	0.0508	0.0240	0.0038	0.0978
`gender`	0.5334	0.3038	-0.0620	1.1288
`educ`	0.0630	0.1214	-0.1749	0.3010
`party5pt`	0.0373	0.0844	-0.1283	0.2028
`X_Ifuncat13_2`	-0.3631	0.5386	-1.4187	0.6926
`X_Ifuncat13_3`	-0.1172	0.5860	-1.2658	1.0314
`X_Ifuncat13_4`	1.5814	0.4612	0.6774	2.4854
`X_Ifuncat13_5`	-0.3687	0.5001	-1.3489	0.6116
`X_Ifuncat13_6`	1.1478	0.5385	0.0923	2.2033
`X_Ifuncat13_7`	1.7630	0.4336	0.9131	2.6128
`X_Ifuncat13_8`	0.9254	0.4965	-0.0478	1.8986
`X_Ifuncat13_9`	0.9789	0.4844	0.0295	1.9284

A Hierarchical Model of Lobbying Influence in the US States, Results

Parameters	Post.Mean	Post.SE	95%.Lower.CI	95%.Upper.CI
`X_Ifuncat13_10`	0.4335	0.4608	-0.4698	1.3367
`X_Ifuncat13_11`	-0.0547	0.4239	-0.8856	0.7761
`X_Ifuncat13_12`	-0.5667	0.5703	-1.6846	0.5512
`timegroupsmed`	1.0671	0.3536	0.3741	1.7602
`timemedXcont`	-0.0602	0.1353	-0.3254	0.2050
`gov6099all`	0.0189	0.0063	0.0066	0.0312
`totreg`	0.0006	0.0007	-0.0008	0.0021
`nonprofs.per10k`	0.0000	0.0000	-0.0000	0.0000
τ	0.0774	0.0042	0.0691	0.0857
τ_α	3.0344	1.3206	0.4460	5.6228

A Hierarchical Model of Lobbying Influence in the US States, Convergence Plots



Using `rjags` from `R` (same model)

```
# CHECK CONVERGENCE
sink(asap2.diags)
superdiag(asap2.mcmc)
sink()

# GET THE DEVIANCE AND THE DIC
asap2.dic <- dic.samples(asap2.model, n.iter=2500, type="pD")
Mean deviance:  3861
penalty 34.2
Penalized deviance: 3895
```

Using rjags from R, Repeat Steps for Null Model

```
asap.null.rjags  <- function()  {
for (i in 1:SUBJECTS) {
    nu[i] <- alpha[state.id[i]]
      + beta[1]*contracting[i] + beta[2]*gov.influence[i] + beta[3]*leg.influence[i]
      + beta[4]*elect.board[i] + beta[5]*years.tenure[i]  + beta[6]*education[i]
      + beta[7]*party.ID[i]    + beta[8]*category2[i]     + beta[9]*category3[i]
      + beta[10]*category4[i]  + beta[11]*category5[i]    + beta[12]*category6[i]
      + beta[13]*category7[i]  + beta[14]*category8[i]    + beta[15]*category9[i]
      + beta[16]*category10[i] + beta[17]*category11[i]   + beta[18]*category12[i]
      + beta[19]*med.time[i]   + beta[20]*medt.contr[i]
    mu[i] <- alpha[state.id[i]]
    grp.influence[i] ~ dnorm(mu[i],tau.y)
    for (j in 1:STATES) {
      eta[j] <- gamma[1]*gov.ideology[j]+gamma[2]*lobbyists[j]+gamma[3]*nonprofits[j]
      alpha[j] ~ dnorm(0,tau.alpha)
    }
    :
}
```

Using `rjags` from `R` (same model)

```
# SAVE MODEL TO A FILE
write.model(asap.null.rjags, "Article.JPART/asap.null.rjags")

# RUN THE SAMPLER AND COLLECT coda SAMPLES
asap.null.out <- jags(data=asap.jags.list, inits=asap.inits, asap.params, n.iter=5000,
    model="Article.JPART/asap.null.rjags", DIC=TRUE)

# USE THE autojags() FUNCTION TO RUN UNTIL G&R < 1.1
asap.null.out2 <- autojags(asap.null.out)

# OR update() TO RUN LONGER
asap.null.out3 <- update(asap.null.out, n.iter=200000)
asap.null.mcmc3 <- mcmc(asap.null.out3)

# CHECK CONVERGENCE
superdiag(asap.null.mcmc3)

# GET THE DEVIANCE AND THE DIC
asap.null.dic <- dic.samples(asap.null.model, n.iter=2500, type="pD")
```

Deviance Comparison

- Three models:

Model	Deviance	Difference	DF	tail value
Null	3963.8	103.7	24	6.9787e-12
Estimated	3861.1			
Saturated	0.0	3861.1	689	<1.0e-300

- This indicates that our model is statistically distinct from both the null model and the saturated model, meaning that we have substantial progress away from the simplest modeling approach but we still have unexplained variance relative to a fully saturated specification.

Exercise 13: Bureaucracy Data

- Rerun the JPART model modifying the covariates.
- Also the corresponding null model.
- Compare results graphically. Be creative.

What Is the *Effective Sample Size* In MCMC?

- Notice that the `JAGS` output gave `n.eff` for each parameter.
- This is an estimate of the equivalent number of *independent* samples for that parameter from the chain.
- First note that the autocorrelation function (ACF) or a series of length n, and lag k is the sum of $n - k$ correlations according to:

$$\rho_k = \sum_{i=1}^{n-k}(x_i - \bar{x})(x_{i+k} - \bar{x}) / \sum_{i=1}^{n}(x_i - \bar{x})^2.$$

- ESS is then calculated by

$$\texttt{n.eff} = \frac{M}{1 + 2\sum_{k=1}^{\infty}\rho_k},$$

  where M is the length of the MCMC run.
- ESS affects the Monte Carlo standard error: how much sampling error is due to MCMC rather than independent Monte Carlo samples: `TimeseriesSE` in the `JAGS` output.

Comments on Burn-in

- The burn-in period is the initial time that is considered to be pre-stationarity.
- We run the chain for some time after the starting point and throw away the values.
- Convergence assessment is essential (diagnostics to come).
- It pays to be conservative in deciding the length of the burn-in period.
- There is no "golden rule" here.
- There are now many users of `JAGS`, but apparently some of them pay little attention to issues of convergence beyond mechanical testing.
- Since MCMC is specifically used to produce marginal distributions, it is predictable that practitioners would only worry marginally.

MCMC Convergence

- Empirical summaries from a given MCMC analysis are not reliable until the chain has reached its stationary distribution and had time to sufficiently mix throughout.
- Until X_t converges at time t ($X \sim \pi$), it is not possible to rely upon the effect of any variant of the CLT.
- Therefore it is necessary to convince yourself and your readers that the Markov chain has mixed throughout its stationary distribution.
- The good news:
  - All ergodic Markov chains are guaranteed to converge asymptotically.
  - It is often quite easy to determine if a given chain has *not* converged.
  - We have lots of tests.

MCMC Convergence (cont.)

- Convergence monitoring:
  - Stop the chain intermittently and apply empirical diagnostics for nonconvergence.
  - A somewhat subjective, ad-hoc process.
  - **Important principle**: these are indicators of *nonconvergence*. So failing to find evidence of nonconvergence with these procedures is not evidence of convergence.
  - Careful practitioners will use more than one tool here.
  - The `R` package **`superdiag`** has the function **`superdiag`** that runs all of commonly used diagnostic tests that we will discuss (and one more).

Example: A Normal-Hierarchical Model of Cold War Military Personnel

- Proportional changes in military personnel for nine east bloc countries (Warsaw Pact plus two) during the period from 1948 to 1983, an interval that covers the height of the Cold War.
- These are collected by Faber (1989, ICPSR-9273) for 78 countries total and include covariates for military, social, and economic conditions.
- Data: Y_{ij} are proportional changes in military personnel for country i at time period j and X_j is the index of the year.
- Specification:

$$
\begin{aligned}
Y_{ij} &\sim \mathcal{N}(\alpha_i + \beta_i X_j, \tau_c) \\
\alpha_i &\sim \mathcal{N}(\alpha_\mu, \alpha_\tau) \\
\beta_i &\sim \mathcal{N}(\beta_\mu, \beta_\tau) \\
\alpha_\mu &\sim \mathcal{N}(1, 0.001) \qquad\qquad \alpha_\tau \sim \mathcal{G}(0.001, 0.001) \\
\beta_\mu &\sim \mathcal{N}(0, 0.001) \qquad\qquad \beta_\tau \sim \mathcal{G}(0.001, 0.001) \\
\tau_c &\sim \mathcal{G}(0.001, 0.001),
\end{aligned}
$$

Example: A Normal-Hierarchical Model of Cold War Military Personnel (cont).

- 1000 iterations of the Markov chain are run and disposed of. The monitoring is turned on for all five nodes and then an additional 100,000 iterations are run.
- Results:

Table 1: Posterior Summary, Military Personnel Model

	Mean	Standard Error	Median	95% HPD
α_μ	9.5502	3.7934	9.7829	[1.4639: 16.4363]
β_μ	-0.0053	0.0192	-0.0053	[-0.0439: 0.0330]
α_τ	0.0150	0.0131	0.0108	[0.0026: 0.0511]
β_τ	398.4455	199.3431	364.7520	[109.3254:875.0422]
τ_c	8.1753	0.6665	8.1613	[6.9151: 9.5301]

- It turns out that although this is hierarchical conjugate setup, that problems in the data cause the chain to be slow mixing. Have we run it long enough?

Correlation and Autocorrelation

- High correlation *between* the parameters of a chain tends to give slow convergence.
- Whereas high correlation within a single parameter (autocorrelation) chain leads to slow mixing and possibly individual *nonconvergence* to the limiting distribution.
- Traceplots are more informative if the burn-in period is omitted.
- Things to look for in traceplots: *trends* and *snaking*.
- Autocorrelation: or a series of length n, the lag k autocorrelation is the sum of $n-k$ correlations according to:

$$\rho_k = \sum_{i=1}^{n-k}(x_i - \bar{x})(x_{i+k} - \bar{x}) / \sum_{i=1}^{n}(x_i - \bar{x})^2$$

- Usually its not necessary to look beyond lag 50.
- Autocorrelation problems can often be helped with reparameterization.

Correlation and Autocorrelation (cont.)

- `BOA` and `CODA` give autocorrelation graphs as well as correlation matrices.
- Military Personnel:

	Within-Chain Correlations				Cross-Chain Correlations			
	Lag 1	Lag 5	Lag 10	Lag 50	β_μ	α_τ	β_τ	τ_c
α_μ	**0.2428**	**0.2210**	**0.2251**	**0.2212**	-0.0203	**0.2745**	0.0054	-0.0953
β_μ	0.0030	0.0029	0.0013	0.0023		-0.0131	0.0084	0.0095
α_τ	**0.5813**	**0.5392**	**0.5367**	**0.5327**			0.0005	**-0.1388**
β_τ	0.0896	0.0110	0.0066	0.0039				-0.0077
τ_c	0.0847	0.0544	0.0623	0.0461				

Geweke Time-Series Diagnostic

- General idea: test based on comparing some proportion of the early era of the chain after the burn-in period with some nonoverlapping proportion of the late era of the chain.
- This suggest a difference of means test using an asymptotic approximation for the standard error of the difference.
- Values that are atypical of a standard normal distribution provide evidence that the two selected portions of the chain differ reasonably (in the first moment).

Geweke Time-Series Diagnostic (cont.)

- Preselect two nonoverlapping window proportions, one early in the chain and one later in the chain: $\boldsymbol{\theta}_1$ of length n_1 and $\boldsymbol{\theta}_2$ of length n_2 along with some function of interest $g()$.
- Diagnostic is given by:

$$G = \frac{\bar{g}(\boldsymbol{\theta}_1) - \bar{g}(\boldsymbol{\theta}_2)}{\sqrt{\frac{s_1(0)}{n_1} + \frac{s_2(0)}{n_2}}}.$$

  and $s_1(0)$ and $s_2(0)$ are standard errors from the symmetric spectral density functions: the uncorrelated contribution from the individual values to the total variance .
- Typical test is for $g()$ to be the mean:

$$\bar{g}(\theta_1) = \sum_{i=1}^{n_1} g(\theta_i)/n,$$

$$\bar{g}(\theta_2) = \sum_{i=1}^{n_2} g(\theta_i)/n,$$

Geweke Time-Series Diagnostic (cont.)

- Geweke suggests using the ratios $n_1/n = 0.1$ and $n_2/n = 0.5$.
- If these proportions are held fixed as the chain grows in length, then the central limit theorem applies and G converges in distribution to standard normal.
- Geweke's idea is that more can be learned by one very long chain since it will end up exploring areas where humans might not think to send it.
- Unfortunately the window proportions can greatly affect the value of G, and it is therefore important not to only use the defaults: (0.1/0.5).
- Military personnel model (yes, with only the defaults):

	α_μ	β_μ	α_τ	β_τ	τ_c
z-statistic	**5.0004**	-1.6375	**4.4763**	-0.4826	**2.0621**
p-value	0.0001	0.1015	0.0001	0.6294	0.0392

Gelman and Rubin's Multiple Sequence Diagnostic

- An ANOVA type test based on multiple chains.
- Steps:
  1. Run $m \geq 2$ chains of length $[2n]$ from overdispersed starting points, $(1), (2), \ldots, (m)$:

$$\theta_{(1)}^{[0]}, \theta_{(1)}^{[1]}, \ldots, \theta_{(1)}^{[2n-1]}, \theta_{(1)}^{[2n]}$$
$$\theta_{(2)}^{[0]}, \theta_{(2)}^{[1]}, \ldots, \theta_{(2)}^{[2n-1]}, \theta_{(2)}^{[2n]}$$
$$\vdots$$
$$\theta_{(m)}^{[0]}, \theta_{(m)}^{[1]}, \ldots, \theta_{(m)}^{[2n-1]}, \theta_{(m)}^{[2n]},$$

     The starting points should be determined by overdispersing around suspected or known modes. Discard the first n chain iterations.

Gelman and Rubin's Multiple Sequence Diagnostic (cont.)

- Continuing steps:

2. For each parameter of interest calculate the following:

   - **Within chain variance:**

$$W = \frac{1}{m(n-1)} \sum_{j=1}^{m} \sum_{i=1}^{n} (\theta_{(j)}^{[i]} - \bar{\theta}_{(j)})^2$$

     where $\bar{\theta}_{(j)}$ is the mean of the n values for the j^{th} chain.

   - **Between chain variance:**

$$B = \frac{n}{m-1} \sum_{j=1}^{m} (\bar{\theta}_{(j)} - \bar{\bar{\theta}})^2$$

     where $\bar{\bar{\theta}}$ is the grand mean (mean of means since each subchain is of equal length).

   - **Estimated variance:**

$$\widehat{\text{Var}}(\theta) = (1 - 1/n)W + (1/n)B.$$

Gelman and Rubin's Multiple Sequence Diagnostic (cont.)

- Continuing steps:

  3. Calculate he convergence diagnostic, a single scalar value, called the *estimated scale reduction* (or shrink factor):

$$\sqrt{\widehat{R}} = \sqrt{\frac{\widehat{\text{Var}}(\theta)}{W}}.$$

- Values of $\sqrt{\hat{R}}$ near 1 are evidence that the m chains are all operating on the same distribution (in practice values less than roughly 1.1 or 1.2 are acceptable).

- Unless one is very worried about multimodality of the prior, the number of separate chains need only be about 5 to 10.

- It is easy to monitor $\sqrt{\hat{R}}$ as the Markov chain runs and move on to other diagnostics when we are happy.

Gelman and Rubin's Multiple Sequence Diagnostic Notes

- Underlying principle:
  - Before convergence W underestimates total posterior variation in θ because the chains have not fully explored the target distribution.
  - W is therefore based on smaller differences early in the chain.
  - Also before convergence $\widehat{\text{Var}}(\theta)$ overestimates total posterior variance because the starting points are intentionally overdispersed relative to the target.
  - Once the chains have converged, the difference should be incidental since the chains are exploring the same region and are therefore overlapping.

Gelman and Rubin's Multiple Sequence Diagnostic Notes)

- Problems/Challenges:
  - It is not always easy to obtain suitably overdispersed starting points, since determining their position requires some knowledge of the target distribution to begin with.
  - This is especially a problem in higher dimensions.
  - Actually this point is critical since the test relies upon the underdispersed/overdispersed distributional contrast. EM can help!
  - G&R also suggest starting with a mixture of normals centered at known nodes and improving the quality of density estimate with importance sampling.
  - Normal assumptions may not always be supportable (somewhat mitigated by the Brooks-Gelman modification [1998]; accommodating finite-sample variability in the variance calculations for B and W) by calculating $\sqrt{\hat{R}}$ for α trimmed intervals.

The Gelman and Rubin Diagnostic for the Military Personnel Model

- 5 Markov chains run with overdispersed starting points determined by gridding.
- 50,000 values recorded using the convention of using only the second half the chain.
- The Brooks and Gelman correction also gives a multivariate summary, which is 5.5569 here, giving evidence nonconvergence in a single dimension can dominate this omnibus assessment of model quality.

	α_μ	β_μ	α_τ	β_τ	τ_c
$\sqrt{\hat{R}}$	1.9593	1.0120	1.0735	1.0001	1.0131
$\text{Corrected}_{0.500}$	**2.1540**	1.0122	**1.2105**	1.0001	1.0132
$\text{Corrected}_{0.975}$	**3.6935**	1.0333	**1.6166**	1.0003	1.0362

Heidelberger and Welch Diagnostic

- Approach originally applied in OR for individual parameters in single chains.
- Based on a Brownian Bridge (Wiener Process) over the interval $[0, 1]$, in which the starting and stopping points are tied to the endpoints 0 and 1 and the "string" in between varies.
- Like the Geweke diagnostic, it has an inherently time-series orientation and uses the spectral density estimate.
- General procedure:
  1. Specify a number of iterations to consider N, an accuracy (ϵ), and an alpha level for the test.
  2. The null hypothesis is that the chain is currently in the stationary distribution and the test starts with the full set of iterations. If the test rejects the null, the first 10% of the iterations are discarded and the test is run again.
  3. This continues until either 50% of the data have been dismissed or the test fails to reject the null with the remaining iterations.
  4. If some proportion of the data are found to be consistent with stationarity, then the *halfwidth* analysis part of the diagnostic is performed.

Heidelberger and Welch Diagnostic (cont.)

- Some details:
  - T = the total length of the "accepted" chain after the discards.
  - $s \in [0{:}1]$ = the test chain proportion.
  - $T_{\lfloor sT \rfloor} = \sum_{i=1}^{\lfloor sT \rfloor} \theta_i$ =the sum of the chain values from one to the integer value just below sT.
  - $\lfloor sT \rfloor \bar{\theta}$ = the chain mean times the integer value just below sT.
  - $s(0)$ = the spectral density of the chain.
- Using these quantities, for any given s, we can construct the test statistic:

$$B_T(s) = \frac{T_{\lfloor sT \rfloor} - \lfloor sT \rfloor \bar{\theta}}{\sqrt{T s(0)}},$$

which is the Cramér-von Mises test statistic for sums as cumulative values scaled by the spectral density.

Heidelberger and Welch Diagnostic for the Military Personnel Model

- Using the standard defaults in `BOA` ($\epsilon = 0.1$, $\alpha = 0.05$), the H-W test gives the following output:

	Stationarity Test	Keep	Discard	C-von-M	Halfwidth Test	Mean	Halfwidth
alpha.mu	**failed**	20000	30000	55.3424033	passed	10.78567427	0.133312358
beta.mu	**failed**	20000	30000	5.8618976	passed	-0.00570531	0.000238181
tau.alpha	**failed**	20000	30000	16.2614855	passed	0.02185464	0.000795133
tau.beta	passed	50000	0	0.1940956	passed	398.44545696	1.949360870
tau.c	**failed**	20000	30000	44.6781527	passed	8.06357154	0.013199792

CODA and BOA for Post-BUGS Analysis

- CODA ("Convergence Diagnostics and Output Analysis") by Martyn Plummer.
- BOA ("Bayesian Output Analysis") by Brian Smith.
- Both free packages that run under R, and are downloaded at CRAN.
- Both provide a rich set of diagnostic tools, including graphics.
- Both can be a little quirky, but are very useful.

Using CODA

```
> install.packages(pkgs="coda",lib=".Rlib")
> codamenu()
CODA startup menu
1:Read BUGS output files
2:Use an mcmc object
3:Quit
Selection: 1
Enter BUGS output filenames, separated by return key
(leave blank to exit)
1: Article.P-Agent/exec13.4
Abstracting k[1] ... 500000 valid values
Abstracting k[2] ... 500000 valid values
Abstracting k[3] ... 500000 valid values
Abstracting k[4] ... 500000 valid values
Abstracting sigma ... 500000 valid values
Abstracting theta[1] ... 500000 valid values
Abstracting theta[2] ... 500000 valid values
Abstracting theta[3] ... 500000 valid values
```

(and so on...)

Using BOA

```
library(boa)
boa.menu()

Bayesian Output Analysis Program (BOA)
Version 1.0.0 for UNIX R
Copyright (c) 2001 Brian J. Smith <brian-j-smith@uiowa.edu>

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License or any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

For a copy of the GNU General Public License write to the Free
Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
```

```
MA  02111-1307, USA, or visit their web site at
http://www.gnu.org/copyleft/gpl.html

NOTE: if the menu unexpectedly terminates, type "boa.menu(recover= TRUE)" to
restart and recover your work

BOA MAIN MENU
*************
1:File     >>
2:Data     >>
3:Analysis >>
4:Plot     >>
5:Options  >>
6:Window   >>

FILE MENU
=========
1:Import Data         >>
2:Load Session
3:Save Session
4:Return to Main Menu
```

```
5:Exit BOA
Selection: 1

IMPORT DATA MENU
----------------
1:BUGS Output File
2:Flat ASCII File
3:Data Matrix Object
4:View Format Specifications
5:Options...
6:Back
7:Return to Main Menu
Selection: 1

Enter filename prefix without the .ind or .out extension [Working Directory: ""]
1: Article.P-Agent/exec.short
Read 1 items
Read 18 records

Read 180000 records
+++ Data successfully imported +++
```

```
IMPORT DATA MENU
----------------
1:BUGS Output File
2:Flat ASCII File
3:Data Matrix Object
4:View Format Specifications
5:Options...
6:Back
7:Return to Main Menu
Selection: 7

BOA MAIN MENU
*************
1:File     >>
2:Data     >>
3:Analysis >>
4:Plot     >>
5:Options  >>
6:Window   >>
Selection: 3
```

```
ANALYSIS MENU
=============
1:Descriptive Statistics  >>
2:Convergence Diagnostics >>
3:Options...
4:Return to Main Menu
Selection: 1

DESCRIPTIVE STATISTICS MENU
---------------------------
1:Autocorrelations
2:Correlation Matrix
3:Highest Probability Density Intervals
4:Summary Statistics
5:Back
6:Return to Main Menu
Selection: 1
Selection: 3

HIGHEST PROBABILITY DENSITY INTERVALS:
```

```
=====================================

Alpha level = 0.05

Chain: Article.P-Agent/exec.short
---------------------------------

          Lower Bound Upper Bound
k[1]        -10.05000    -0.88130
k[2]         -5.93900     1.43900
k[3]         -1.82500     4.34400
k[4]          4.33500    12.20000
sigma         3.21600     8.75000
tau           0.00945     0.06781
theta[10]     0.22970     1.33000
theta[11]    -1.08300     0.05925
theta[12]     0.62890     3.93200
theta[1]     -4.13900     2.87200
theta[2]     -1.86500     0.55190
theta[3]     -0.83480     0.17190
theta[4]      0.36910     2.05700
```

```
theta[5]      0.28810     3.25600
theta[6]     -3.68100    -1.61800
theta[7]      0.54550     2.15500
theta[8]     -1.66400    -0.35180
theta[9]     -0.14270     0.86110

Selection: 4

SUMMARY STATISTICS:
===================

Bin size for calculating Batch SE and (Lag 1) ACF = 50

Chain: Article.P-Agent/exec.short
---------------------------------

                Mean          SD     Naive SE     MC Error    Batch SE
k[1]      -5.59806203 2.42271566 0.0242271566 0.1223327834 0.166725341
k[2]      -2.42940786 1.94031614 0.0194031614 0.0968573426 0.132938124
k[3]       1.35981060 1.60391981 0.0160391981 0.0778199101 0.108735974
k[4]       8.40854860 1.96700713 0.0196700713 0.1001023043 0.134728021
```

sigma	5.81126050	1.40852457	0.0140852457	0.0750363453	0.097579773
tau	0.03498487	0.01690473	0.0001690473	0.0008895715	0.001164072
theta[10]	0.76229775	0.29844260	0.0029844260	0.0153456628	0.020729837
theta[11]	-0.47943636	0.29944492	0.0029944492	0.0145505332	0.020824001
theta[12]	2.12008306	0.82920162	0.0082920162	0.0403465492	0.049794374
theta[1]	-0.44837512	2.13613478	0.0213613478	0.1104139879	0.150234893
theta[2]	-0.64348071	0.63055720	0.0063055720	0.0309518537	0.038675137
theta[3]	-0.31625976	0.25289936	0.0025289936	0.0131013569	0.017111244
theta[4]	1.11830299	0.44912150	0.0044912150	0.0226997128	0.031597119
theta[5]	1.71032372	0.77381701	0.0077381701	0.0352462425	0.045846571
theta[6]	-2.67945860	0.58558035	0.0058558035	0.0312714378	0.041294397
theta[7]	1.18996756	0.47287189	0.0047287189	0.0219656503	0.033222243
theta[8]	-0.85445191	0.33927737	0.0033927737	0.0181838079	0.023634776
theta[9]	0.34877322	0.24568023	0.0024568023	0.0128376865	0.016650594

	Batch ACF	0.025	0.5	0.975	MinIter	MaxIter	Sample
k[1]	0.9075014	-9.87607500	-5.73300	-0.63469500	1	10000	10000
k[2]	0.8830128	-5.81300000	-2.56200	1.60505000	1	10000	10000
k[3]	0.8511580	-1.88007500	1.40200	4.30900000	1	10000	10000
k[4]	0.8960185	4.43700000	8.49550	12.31000000	1	10000	10000
sigma	0.9565878	3.62395000	5.70100	9.32520000	1	10000	10000

tau	0.9305307	0.01149950	0.03077	0.07616125	1	10000	10000
theta[10]	0.9185856	0.23176000	0.75560	1.33500000	1	10000	10000
theta[11]	0.9200565	-1.03525000	-0.47600	0.16190000	1	10000	10000
theta[12]	0.5477688	0.55450000	2.07700	3.88300000	1	10000	10000
theta[1]	0.9651961	-4.03800000	-0.26025	3.02500000	1	10000	10000
theta[2]	0.6337737	-1.98707500	-0.57785	0.49360000	1	10000	10000
theta[3]	0.8160511	-0.83090000	-0.30690	0.19560000	1	10000	10000
theta[4]	0.9594892	0.41830000	1.04000	2.13605000	1	10000	10000
theta[5]	0.6044240	0.31262750	1.67100	3.31900000	1	10000	10000
theta[6]	0.9783000	-3.80200000	-2.58350	-1.69700000	1	10000	10000
theta[7]	0.9560934	0.52260000	1.09300	2.13900000	1	10000	10000
theta[8]	0.9260070	-1.76300000	-0.77130	-0.38477250	1	10000	10000
theta[9]	0.8242078	-0.08368125	0.32735	0.93360000	1	10000	10000

Exercise 14: Diagnostics for the Bureaucracy Data

- Return to your analysis of the bureaucracy data with `JAGS`.
- In that exercise we did not worry about convergence diagnostics.
- Now run all of the standard convergence diagnostics, either with `superdiag`, `coda`, or `boa`.
- Consider how well your burn-in period worked.